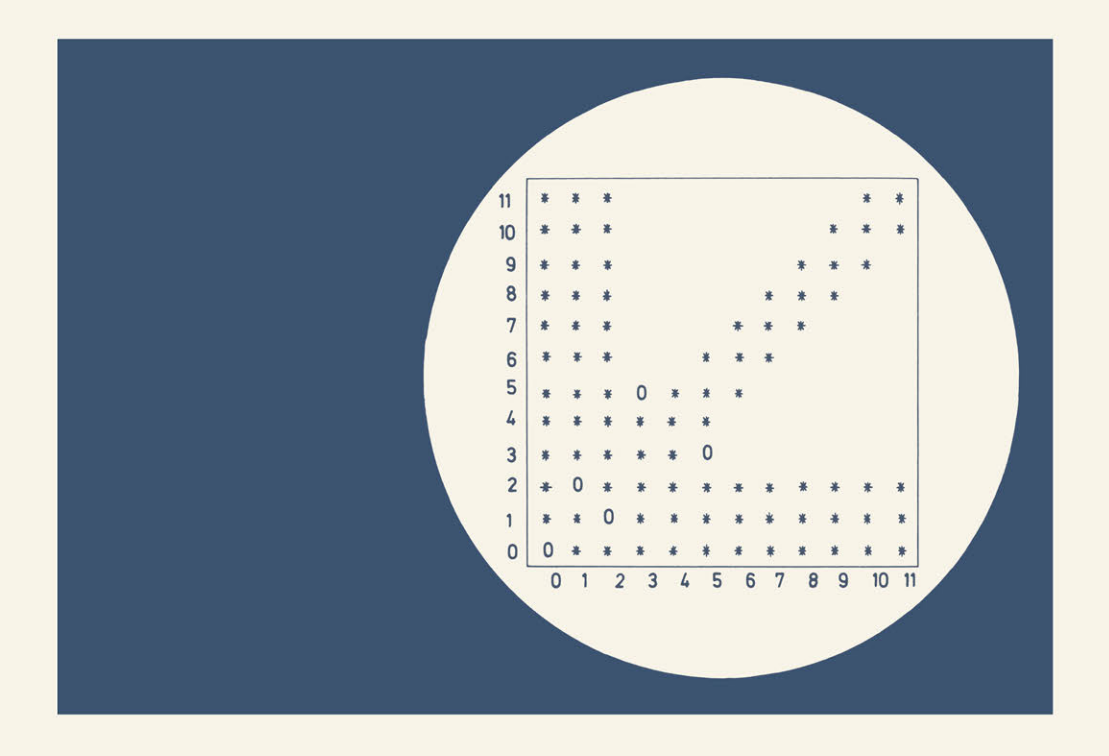
Vieweg Programmbibliothek Mikrocomputer 3

BASIC und Pascal im Vergleich

Graphische Darstellung von Programmablaufplänen
Vor- und Nachteile beim Programmieren
Vergleich bei Spielen



Vieweg Programmbibliothek
Mikrocomputer 3

BASIC und Pascal
im Vergleich

Vieweg Programmbibliothek Mikrocomputer

Herausgegeben von Harald Schumny

Band 1
Graphik-Programme für TRS-80 und HP 9830

Band 2
Iterationen, Näherungsverfahren, Sortiermethoden

BASIC-Programme für
CBM 3032, HP 9830, TRS-80, Olivetti 6060

Band 3
BASIC und Pascal im Vergleich

Band 4
BASIC-Anwenderprogramme

Vieweg Programmbibliothek
Mikrocomputer Band 3

Harald Schumny (Hrsg.)

BASIC und Pascal im Vergleich



Springer Fachmedien Wiesbaden GmbH

CIP-Kurztitelaufnahme der Deutschen Bibliothek

BASIC und PASCAL im Vergleich / [d. Autoren d. Bd. Karl Achilles ...].
(Vieweg-Programmbibliothek Mikrocomputer; Bd. 3)
ISBN 978-3-528-04224-0 ISBN 978-3-663-14221-8 (eBook)
DOI 10.1007/978-3-663-14221-8

NE: Achilles, Karl [Mitverf.]; GT

Die Autoren des Bandes

Karl Achilles
Neuenstraße 2, 2805 Stuhr 1
Studienrat an der KGS Stuhr-Brinkum

Rüdeger Baumann
In den Stuken 16, 2120 Lüneburg
Studiendirektor am Gymnasium Lüneburg

Dietmar Herrmann
Lärchenstraße 20, 8011 Anzing
Lehrer

Prof. Dr.-Ing. *Wolfgang Schneider*
Wanneweg 1, 3302 Cremlingen
Fachhochschullehrer an der FH Braunschweig-Wolfenbüttel

1983

Ursprünglich erschienen bei Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1983

ISBN 978-3-528-04224-0

Inhaltsverzeichnis

Einführung

BASIC ist heute so etwas wie eine Standard-Programmiersprache, und für viele private und berufliche Computer-Verwender ist BASIC die als erste gelernte Sprache. Das liegt vor allem an der leichten Erlernbarkeit – selbst Anfänger können bereits nach wenigen Stunden eigene Programme schreiben. Praktische Gründe für die Dominanz von BASIC sind aber auch die Dialogfähigkeit, die interaktives Arbeiten am Computer ermöglicht (Mensch-Maschine-Dialog), und die Tatsache, daß die am meisten verbreiteten Tischcomputer nur BASIC „verstehen".

Häufig hört man von erfahrenen Programmierern, BASIC weise eine Reihe ernst zu nehmender Mängel und Nachteile auf, vor allem

- BASIC erlaubt keine strukturierte Programmierung, verleitet vielmehr zu extensiven Verzweigungen und Verschachtelungen („Spaghetti"-Stil);
- es sind nur globale Variablen möglich (nur solche, die für das ganze Programm gelten), lokale Variablen z. B. in Unterprogrammen können nicht definiert werden;
- symbolische Adressierung ist in der Regel nicht möglich, Variablennamen sind oft viel zu kurz (meist nur 2 Zeichen);
- das Aneinanderhängen mehrerer Programme (chain, append oder merge) durch Zuladen vom Massenspeicher ist normalerweise nicht möglich;
- WHILE-Schleifen, CASE-Strukturen und IF-THEN-ELSE werden nur äußerst selten geboten.

Das stimmt natürlich alles. Es sollte trotzdem immer ehrlich abgewogen werden, ob man auf all diese fehlenden Möglichkeiten nicht auch verzichten kann. Die weite Verbreitung von BASIC läßt den Schluß zu, daß in sehr vielen Anwendungsfällen tatsächlich darauf verzichtet werden kann.

Es gibt jedoch in professionellen Anwendungsbereichen sehr schnell Fälle, die den Programmierer sofort mit den aufgeführten Mängeln konfrontieren. Das wissen natürlich auch Computerhersteller. Und so gibt es inzwischen einige Tischcomputer auch unter 10 000,– Mark, die mit erweiterten BASIC-Versionen arbeiten, oft als enhanced oder extended BASIC bezeichnet. Manchmal werden sogar BASIC-Dialekte angeboten, die eine Quasi-Strukturierung zumindest beim Ausdrucken der Anweisungslisten ermöglichen.

Auf der anderen Seite wird man zunehmend mit Pascal konfrontiert. Und es kann gesagt werden, daß die Programmiersprache Pascal die eben bei BASIC aufgezeigten Mängel nicht aufweist. Es gibt aber andere Nachteile:

- Pascal ist nicht ganz so leicht erlernbar wie BASIC;
- interaktives Arbeiten ist nicht so ohne weiteres möglich, weil Pascal kompilierend umgesetzt wird;
- die äußerst bequemen Möglichkeiten von Stringmanipulationen in der Programmiersprache BASIC existieren in Pascal nicht;
- Ein-/Ausgabeoperationen werden von der Sprachendefinition her nicht unterstützt.

Es gibt aber entscheidende Vorteile in Pascal. Vor allem zu nennen sind die Möglichkeiten, verschiedene Variablentypen mit relativ „langen" Variablennamen zu definieren, und die Tatsache, daß man sozusagen zu strukturierter Programmierung gezwungen wird.

In diesem Band der Programmbibliothek haben wir eine Reihe von Beiträgen zusammengestellt, in denen BASIC und Pascal in sehr eingängiger Weise gegenübergestellt sind. Vorangestellt ist der Aufsatz „Graphische Darstellung von Programmabläufen" von **Wolfgang Schneider**. Darin werden die beiden Möglichkeiten der Programmbeschreibung bzw. -dokumentierung diskutiert, die jeweils gerade den beiden hier verglichenen Programmiersprachen angemessen erscheinen: die konventionellen Programmablaufpläne für „ältere" Sprachen und Struktogramme für Pascal.

Es folgen zwei Beiträge von **Rüdeger Baumann** mit algorithmisch betrachteten Spielen, worin die „Vorzüge der Programmiersprache Pascal gegenüber BASIC" besonders gut herausgestellt werden können. Es sei an dieser Stelle der durch Rüdeger Baumann dem Herausgeber übermittelte Stoßseufzer von Sidonie Trampler (Literaturzitat [17, S. 104] im Beitrag „Ein Spiel — ...") angeführt: „Der Herr behüte uns vor den von Didaktikern zum Zweck des Mathematiklernens erfundenen ‚Spielen'!". Die Spiele für den BASIC/Pascal-Vergleich sind gerade nicht von dieser Sorte.

Ein großer Block mit 10 Beiträgen stammt von **Dietmar Herrmann**. Die einzelnen Beispiele werden anschaulich eingeführt, und es wird gezeigt, daß die verwendeten Algorithmen meist direkt in Pascal programmiert werden können. Für Lösungen mit BASIC müssen häufig umständliche Simulierungen und Berechnungsroutinen geschrieben werden, die sinnvollerweise in der Regel als Unterprogramme aufzurufen sind.

Der letzte Block in diesem Bibliotheksband stammt von **Karl Achilles**. Anhand von fünf Beispielen wird auch dabei aufgezeigt, wo jeweils Vor- oder Nachteile bei der Programmierung in BASIC bzw. Pascal zu sehen sind. Auch hierbei wird deutlich, daß die aufgestellten Algorithmen zumeist direkt eine entsprechende Pascal-Struktur widerspiegeln.

Möglicherweise wird durch solche Veröffentlichungen, wie die hier vorliegende, der Wunsch nach Pascal verstärkt. Es sollte aber nicht vergessen werden, daß BASIC bei Anfängern leichter die Schwellenangst abbauen kann bzw., andersherum, daß durch erstmalige Konfrontation mit Pascal manchmal Abneigungen verstärkt werden können. Für alle in der Prozeßdatenverarbeitung tätigen Anwender bleibt Pascal wegen der fehlenden Ein-/Ausgabe-Strukturen sowieso zweite Wahl, es sei denn, die Assembler-Sprache wird zur Programmierung von Schnittstellentreibern akzeptiert. Wahrscheinlich aber wird dieser spezielle Anwenderkreis eher zu „prozeßgeeigneten" Sprachen tendieren und dann auf beispielsweise Forth stoßen, eine wenigstens 10 Jahre „alte" Sprache, die aber erst in jüngster Zeit auch für Mikrocomputer interessant wird. Oder man gelangt zur neuen „Universalsprache" Ada, die vom US-Verteidigungsministerium (Department of Defense, DoD) kontrolliert und lizenziert wird. (Ada-Compiler dürfen nur so heißen, wenn die DoD-Lizenz vorliegt!)
Wir bleiben aber zunächst noch in der praktischen Gegenwart und stellen im folgenden BASIC und Pascal im Vergleich gegenüber.

Graphische Darstellung von Programmabläufen

von Wolfgang Schneider

Vor der Programmierung eines Problems in einer beliebigen Programmiersprache empfiehlt es sich,

- das Problem aufzubereiten und
- den Programmablauf graphisch darzustellen.

Erst anschließend sollte man, zumindest bei umfangreichen Problemen, zum Schreiben des Primärprogrammes in einer beliebigen Programmiersprache übergehen.

1 PROBLEMAUFBEREITUNG

Zur Problemaufbereitung gehört

- eine vollständige Formulierung der Aufgabe und
- eine Problemanalyse der Aufgabe.

Die Aufgabe ist zunächst vollständig mit allen Randbedingungen in der Umgangssprache zu formulieren. In der darauf folgenden Problemanalyse ist u.a. zu untersuchen,

- ob die Aufgabe überhaupt mit Hilfe einer Datenverarbeitungsanlage (kurz DVA) gelöst werden kann,
- welche alternativen Lösungswege sich für die Aufgabe anbieten und
- welcher der möglichen Lösungswege der günstigste ist.

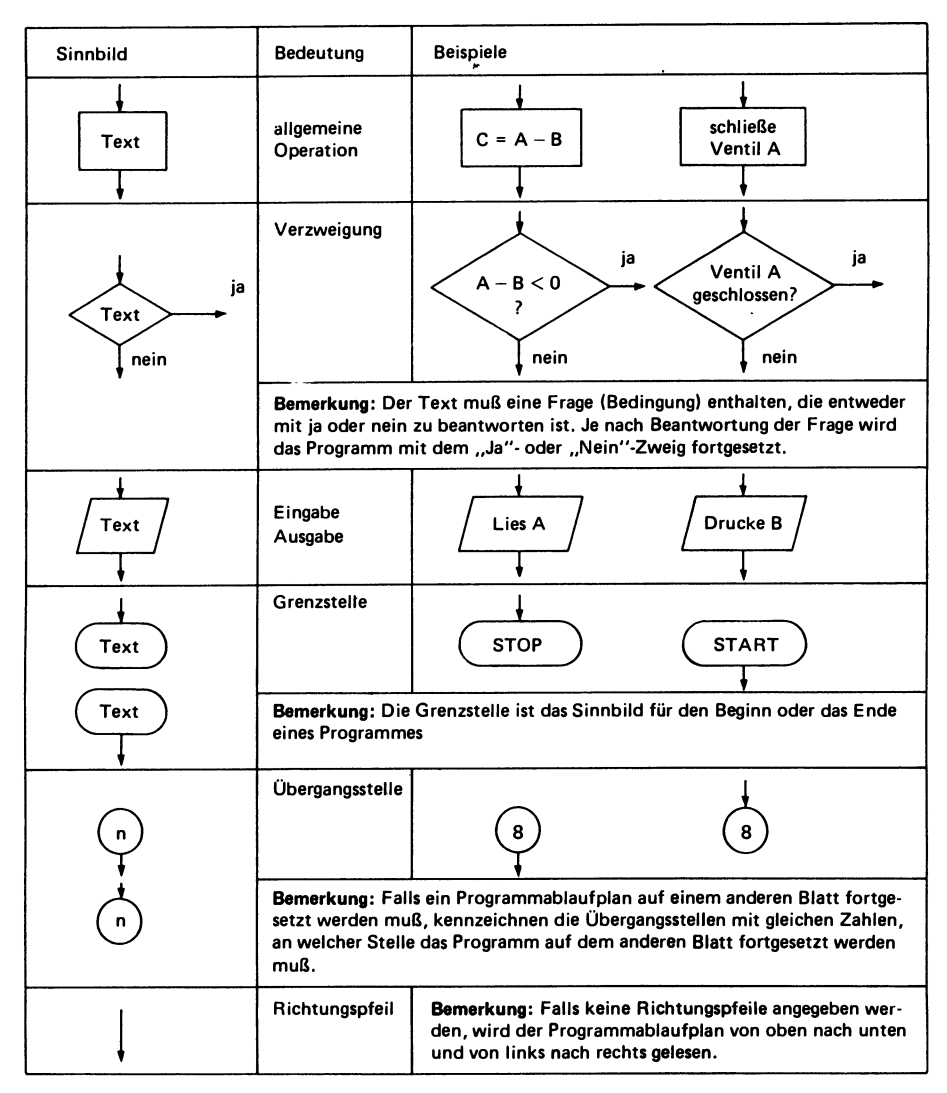
2 DARSTELLUNG VON PROGRAMMABLÄUFEN

Nachdem bei der Problemaufbereitung ein günstig erscheinender Lösungsweg gefunden wurde, empfiehlt es sich vielfach, die einzelnen Schritte zur Lösung des Problems graphisch darzustel-

len. Hier stehen dem Programmierer zwei wichtige Darstellungsweisen zur Verfügung:

- Darstellung mit Hilfe von Programmablaufplänen.
- Darstellung mit Hilfe von Struktogrammen.

Diese beiden Darstellungsweisen sollen im folgenden besprochen und miteinander verglichen werden.



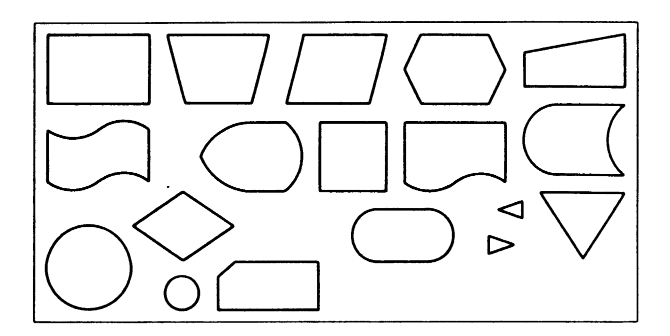
B i l d 1: Nach DIN 66001 genormte Sinnbilder von Programmablaufplänen:

2.1 Programmablaufpläne

Ein Programmablaufplan ist eine graphische Darstellung, die den Arbeitsablauf einer Problemstellung in einzelnen Schritten darstellt.

Die Programmablaufpläne setzen sich aus verschiedenen Sinnbildern zusammen, die nach DIN 66001 genormt sind. Durch Einfügen eines Textes in die Sinnbilder wird die Art der Vorgänge genau spezifiziert. Die Reihenfolge der Vorgänge wird durch Pfeile angedeutet. Für einfache Aufgaben genügt die Kenntnis der in B i l d 1 aufgeführten Sinnbilder.

Das Zeichnen der Programmablaufpläne wird durch Schablonen erleichtert (B i l d 2).



B i l d 2: Beispiel einer Schablone zur Erstellung von Programmablaufplänen

Vorteile bei der Anwendung von Programmablaufplänen:

Der Programmablaufplan erweist sich bei umfangreichen Aufgaben als sehr zweckmäßig. Insbesondere sind folgende Vorteile zu nennen:

- o Der Programmablaufplan verschafft dem Programmierer durch die logische Gliederung einen Überblick über den Gang der Rechnung.

- o Der Programmablaufplan verhindert das Programmieren von "Sackgassen".
  Eine Sackgasse würde sich z.B. in einem Programm ergeben, wenn ein Zweig einer Verzweigung durch Vergeßlichkeit des Programmierers nicht weiter berücksichtigt würde. Wenn bei einer späteren Benutzung des Programms dieser Zweig gewählt wird, so gibt es keine darauffolgende Anweisung. Das Programm endet in einer Sackgasse. In einem Programmablaufplan sind derartige Sackgassen gut zu erkennen und können somit vermieden werden.
- o Der Programmablaufplan bietet ein gutes Verständigungsmittel zwischen einem Spezialisten und einem Programmierer.
  Spezialisten verfügen teilweise über keine ausreichenden Programmierkenntnisse. Ein Programmierer hingegen verfügt nicht immer über die notwendigen Spezialkenntnisse, um programmierbare Regeln aus einer Aufgabenstellung abzuleiten. Hier bietet sich der Programmablaufplan als gemeinsames Verständigungsmittel an.
- o Der Programmablaufplan hilft bei der Fehlersuche von logischen Fehlern.
  Durch die logische Gliederung des Problems in eine Folge von Einzelschritten ist der Programmablaufplan wegen der besseren Übersicht meist besser als das Programm selbst geeignet, logische Fehler im Programmablauf zu finden.
- o Der Programmablaufplan dient zur Dokumentation des Programms.

  Programme sollen auch später, eventuell von anderen Personen, wieder benutzt werden können. Sie müssen sich auf einfache Art darüber informieren können, wie das Programm aufgebaut ist, welcher Lösungsweg gewählt wurde usw. In vielen Fällen kann der Programmablaufplan eine spezielle Programmbeschreibung ersparen.

2.2 Struktogramme

Programmablaufpläne sind als Hilfsmittel zur graphischen Darstellung von Programmen weit verbreitet und auch genormt. Sie beschreiben die logische Struktur eines Algorithmus und den Ablauf, d.h. die Reihenfolge der Schritte zu programmierender Probleme. Die Praxis zeigt jedoch, daß dieses Hilfsmittel den

Programmierer dazu verführt, Programme ohne größere Überlegung an beliebigen Stellen zu verzweigen und mit Hilfe der Übergangsstellen an beliebigen Orten, oft auf anderen Blättern, wieder zusammenzuführen. Dadurch ist ein Programm vielfach nicht mehr in einfache, selbständige Blöcke aufteilbar, d.h. die Strukturierung des Programms wird erschwert oder sogar unmöglich gemacht. Die Strukturierung ist jedoch für umfangreiche Problemstellungen sehr wichtig. Aus diesem Grunde haben NASSI und SHNEIDERMANN eine graphische Darstellungsmethode entwickelt, die einen "Sprung" von einem Punkt zu einem anderen Punkt im Programmablaufplan verhindert. Damit wird gleichzeitig sichergestellt, daß nicht nur der "Programmfluß", sondern auch die "Programmstruktur" deutlich wird, d.h. die Bedeutung der einzelnen Programmteile für den Gesamtablauf wird auf den ersten Blick erkennbar.

Sinnbild	Bedeutung	Erläuterung
Text	*Prozeß* (Aktivitat, Operation)	Das Prozeßsinnbild dient zur Darstellung eines oder mehrerer Befehle wie z.B. • Wertzuweisungen (arithmetische Zuordnungsanweisungen) • Ein- und Ausgabeanweisungen (diese besitzen in Programmablaufplanen ein spezielles Sinnbild) • Unterprogrammaufrufe (diese besitzen in Programmablaufplanen ebenfalls ein spezielles Sinnbild, das allerdings in Bild 1 nicht aufgenommen wurde). Die Form des Prozeßsinnbildes ist rechteckig. Die Große ist frei wahlbar.
Text F T	*Verzweigung* (Entscheidung, Selektion)	Das Verzweigungssinnbild dient zur Darstellung bedingter Verzweigungen mit zwei Alternativen (**Ja/Nein-Entscheidung**). Das Verzweigungssinnbild besteht aus drei Dreiecken. Das mittlere Dreieck (Text) enthalt die Bedingung (Frage), die entweder mit NEIN (F = FALSE) oder JA (T = TRUE) zu beantworten ist. Je nach Beantwortung der Frage wird das Programm mit einem Prozeßsinnbild, das direkt auf das linke Dreieck (F) oder auf das rechte Dreieck (T) folgt, fortgesetzt. Die Große des Verzweigungssinnbildes ist von der jeweiligen Anwendung und dessen Erfordernissen abhangig.
Wiederholungsbed. Rumpf	*Wiederholung* (Schleife, Iteration)	Das Wiederholungssinnbild dient zur Darstellung von Schleifen. Die Wiederholungsbedingung steht links oben im Wiederholungssinnbild. Hier wird z.B. die Zahl der Wiederholungen angegeben oder die Bedingung, unter der die Wiederholung zu beenden ist. Die zu wiederholenden Anweisungen stehen im inneren Rechteck (Rumpf). Der Rumpf kann aus einer Struktur beliebiger Verwicklung bestehen. Eine Verschachtelung von Schleifen ist moglich (weitere Schleifen im Rumpf!).
BEGIN Rumpf END	*Anfang und Ende*	Das Anfang- und Ende-Sinnbild dient zur Darstellung des Beginns oder des Endes von Programmen.

B i l d 3: Grund-Symbole für Struktogramme

Das Lesen und Zeichnen von Struktogrammen ist schnell erlernbar, da im wesentlichen nur 4 Basis-Symbole verwendet werden, die in speziellen Fällen etwas verändert werden. Sie sind zwar nicht genormt, jedoch in der Literatur bislang gleichartig dargestellt worden. Die 4 Basis-Symbole und zwei häufig benutzte Varianten zeigen Bild 3 und Bild 4.

Sinnbild	Bedeutung	Erlauterung
Text / 1 | 2 | 3 | 4	*Mehrfachverzweigung* (Schalter)	Das Mehrfachverzweigungssinnbild dient zur Darstellung bedingter Verzweigungen mit mehr als zwei Alternativen. Das obere Dreieck des Sinnbildes enthält die Fallabfrage (Bedingung), d.h. hier wird angegeben, unter welcher Bedingung zu den einzlnen Fällen (1, 2, ... n) verzweigt wird.
Wiederholungsbed. / Rumpf / Abbruchbed.	*Schleife mit Abbruchbedingung*	Dieses Sinnbild dient zur Darstellung von Schleifen, die unter bestimmten Bedingungen abzubrechen sind. Es gleicht dem normalen Wiederholungssinnbild mit dem Unterschied, daß im Rumpf die Abbruchbedingung aufgenommen ist.

Bild 4: Häufig benutzte Symbolvarianten für Struktogramme

2.3 Beispiele

Beispiel 1

Aufgabenstellung:

Es soll die Summe der ganzen geraden Zahlen von 1 bis 20 gebildet werden. Geben Sie zur Lösung dieses Problems sowohl einen Programmablaufplan als auch ein Struktogramm an.

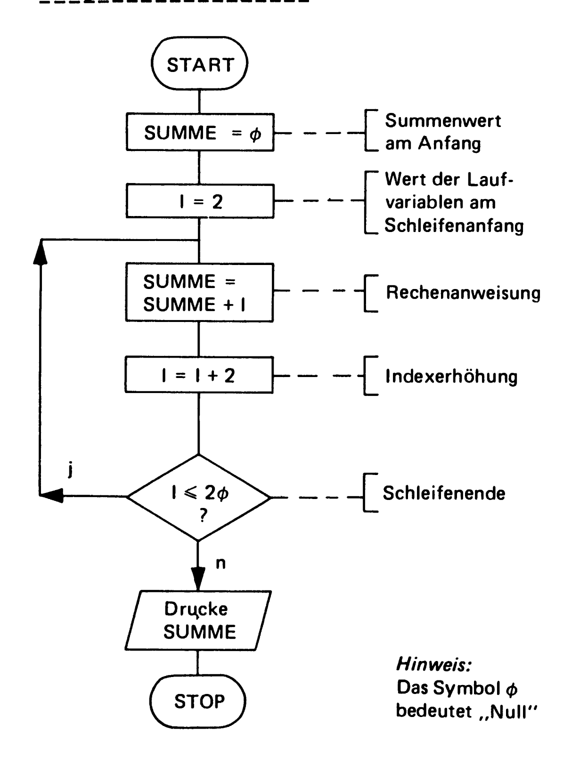
Erklärung:

Zunächst müssen die Anfangswerte gesetzt werden. So wird zunächst der Inhalt der Speicherzelle, in der die Summen nach jedem Schleifendurchlauf abgespeichert werden, Null gesetzt, damit Werte, die vorher möglicherweise in der Speicherzelle standen, das Ergebnis nicht verfälschen können (Summe = Ø).

Außerdem wird der Anfangswert der Laufvariablen I auf den ersten ganzzahligen Wert, der zur Summe beiträgt, festgelegt (I = 2).

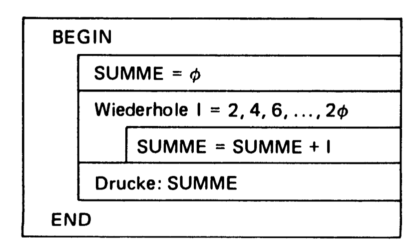
Es erfolgt anschließend die erste Rechnung: SUMME = Ø + 2.
Darauf folgt die erste Indexerhöhung, die sich aus I = I + 2

Programmablaufplan:



mit den vorgegebenen Werten zu I = 2 + 2 = 4 ergibt. Durch eine Abfrage, ob I ≤ 2Ø ist, wird der Index I überprüft. Ist der Wahrheitswert wahr, wird die zweite Rechnung SUMME = 2 + 4 ausgeführt usw. Die Schleife wird so lange durchlaufen, bis I > 20 wird. Dann wurden alle ganzen geradzahligen Zahlen einschließlich 20 aufaddiert und die Summe kann gedruckt werden.

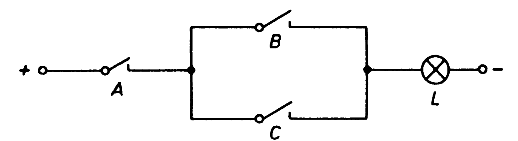
Struktogramm:



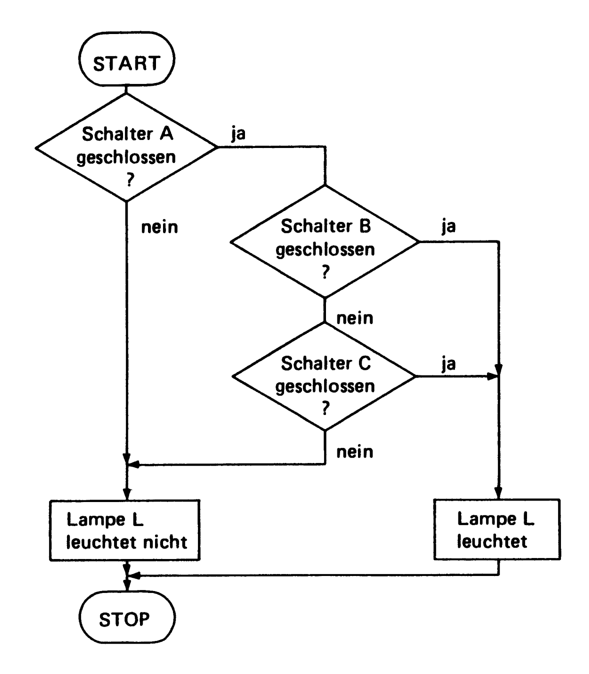
Beispiel 2

Aufgabenstellung:

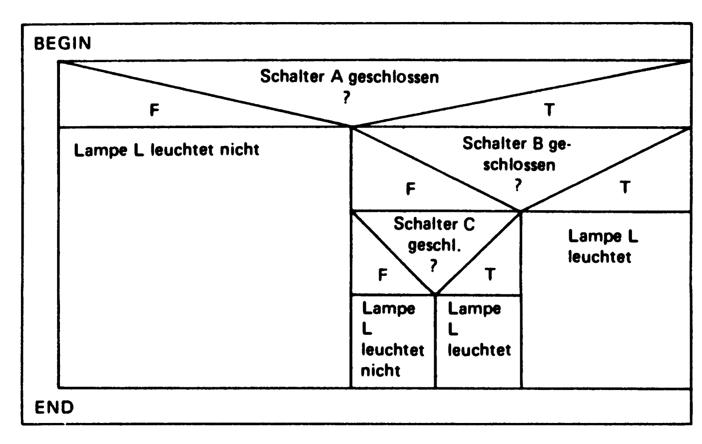
Es soll die Ablaufstruktur folgender Lampenschaltung graphisch mit Hilfe eines Programmablaufplanes und eines Struktogrammes dargestellt werden, die die Lampe zum Leuchten bzw. nicht zum Leuchten bringt.



Programmablaufplan:



Struktogramm:

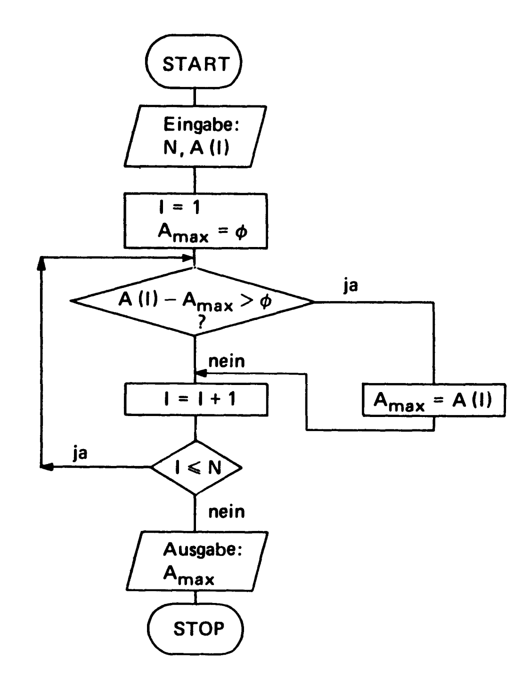


Beispiel 3

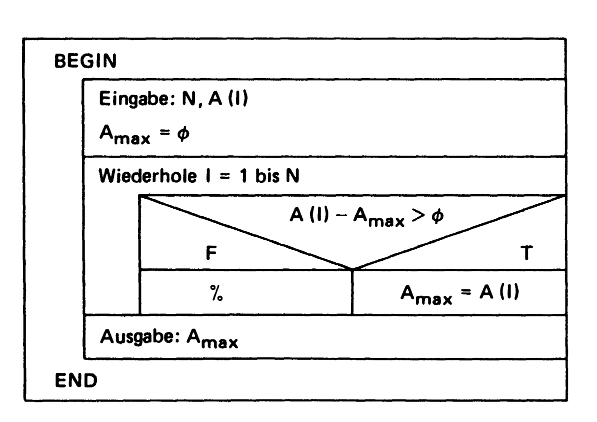
Aufgabenstellung:

Aus einer bestimmten Anzahl N von positiven Zahlen A(I) soll durch einen Suchvorgang die maximale Zahl Amax ermittelt werden. Die Ablaufstruktur soll mit Hilfe eines Programmablaufplanes und eines Struktogrammes dargestellt werden.

Programmablaufplan:



Struktogramm:



Ein Spiel – algorithmisch betrachtet

von Rüdeger Baumann

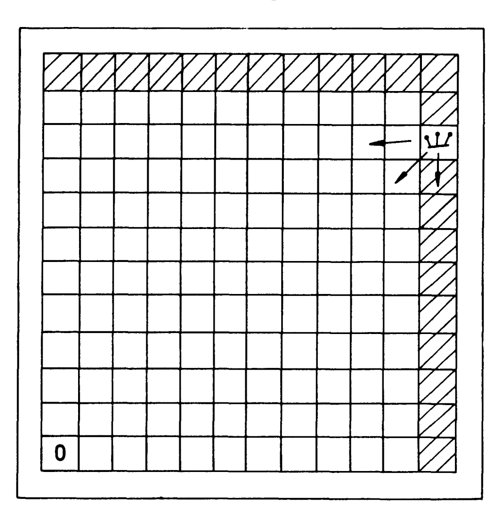
> Warum Spiele? fragen Sie.
> Ich antworte: um die Kunst
> der Erfindung zu vervollkommnen.
>
> LEIBNIZ 1716

Das folgende Beispiel unterbreite ich aus drei Gründen: erstens seines ästhetischen Reizes und der vielfältigen mathematischen Bezüge wegen; zweitens, weil es die gemeinsame algorithmische Struktur aller deterministischen Zweipersonenspiele zeigt; und drittens, weil es Gelegenheit gibt, Vorzüge der Programmiersprache Pascal gegenüber BASIC herauszustellen.

1 WYTHOFFs NIM

Ein schon den alten Chinesen bekanntes, von dem Holländer W.A. Wythoff 1907 neuentdecktes Nim-ähnliches Spiel lautet (in geometrischer Fassung) folgendermaßen:

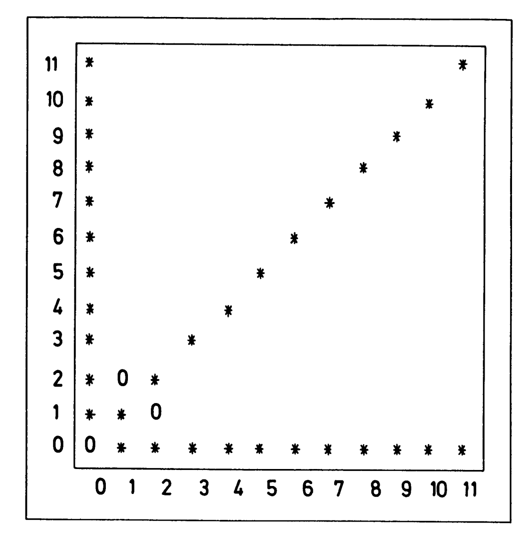
Gegeben ist ein schachbrettartiges Spielfeld beliebiger Größe und eine Damefigur.



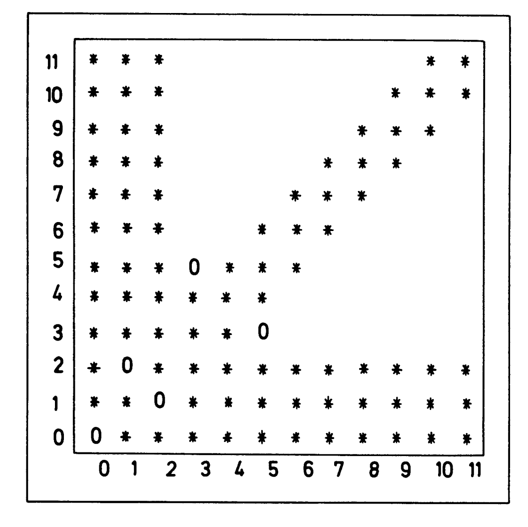
Spieler A setzt die Dame auf eines der Felder der obersten Zeile oder der am weitesten rechts befindlichen Spalte (in der Abbildung schraffiert). Die Dame kann wie beim Schachspiel bewegt werden, aber nur in den Richtungen West, Südwest, Süd

(siehe die Pfeile). B hat den ersten Zug, dann wird abwechselnd gezogen. Wer nicht mehr ziehen kann, weil die Dame im Eck O steht, hat verloren.

Wir stellen sofort fest: Steht die Dame in der Zeile, Spalte oder Diagonale, welche die Null O enthält, kann der am Zug befindliche Spieler unmittelbar gewinnen, indem er die Dame auf das O-Feld zieht. Wir markieren die Felder dieser Reihen (mit jeweils einem Stern *), da wir sie beim Setzen der Dame vermeiden müssen.



Als nächstes nicht markiertes Feld erhalten wir (1;2): es bezeichnet eine Verluststellung für den gerade am Zug befindlichen Spieler. Wie er auch zieht, immer gerät er auf ein mit * markiertes Feld und muß verlieren. Auf diese Weise ermitteln wir, von der Endstellung her aufsteigend, rekursiv die Verluststellungen und belegen sie mit dem Zeichen O.



Es handelt sich um eine Art Siebverfahren (analog zum Sieb des Eratosthenes), das sich folgendermaßen beschreiben läßt:

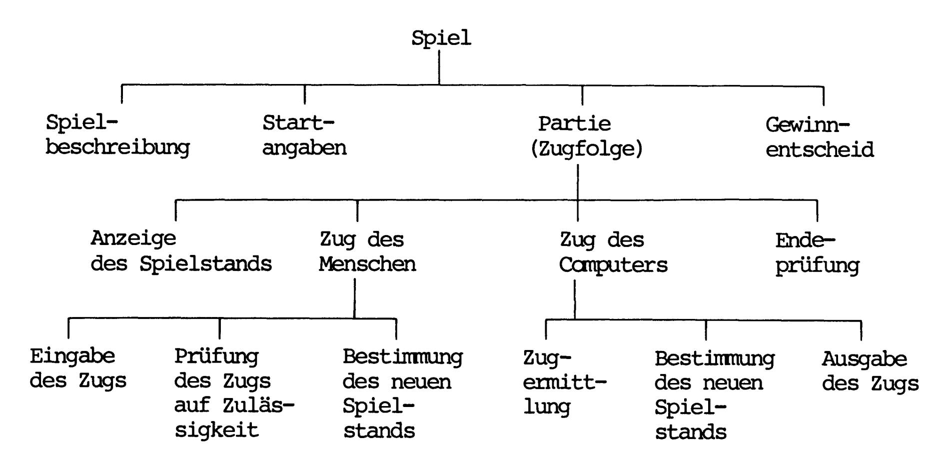
WYTHOFF-SIEB

```
FÜR  x  VON  Ø  BIS  n  WIEDERHOLE
  FÜR  y  VON  Ø  BIS  n  WIEDERHOLE
    WENN  brett(x,y) = Ø  DANN
      gib position x,y aus; (* Verluststellung *)
      markiere alle von  x,y  ausgehenden  waagerechten,
      senkrechten und diagonalen reihen
    ENDE-WENN
  ENDE-WIEDERHOLE
ENDE-WIEDERHOLE
```

Dies ist der Kern einer vom Computer anzuwendenden Strategie: er braucht die Dame lediglich auf die nächstliegende so ermittelte Verluststellung zu ziehen (sofern sie nicht schon auf einer steht; in diesem Fall macht der Computer einen Verlegenheitszug und hofft auf einen Fehler des Gegners).

2 SPIELPROGRAMM

Jedes Wettkampfspiel zwischen zwei Personen (Mensch und Computer) läßt sich gliedern, wie es das folgende Baumdiagramm beschreibt (siehe Schrage [14]):



Die Besonderheiten unseres Spiels kommen erst bei der Ermittlung des Computerzugs zur Geltung. Bevor wir darauf eingehen, wird - gemäß dem Baumdiagramm - die Grobstruktur des Spielprogramms entwickelt; es lautet in der Programmiersprache Pascal:

```
PROGRAM treib-die-dame;
  (* nim-ähnliches strategiespiel, das von den alten chine-
     sen erfunden, und von w.a. wythoff neuentdeckt wurde *)

  CONST
    seitenlänge  = 5Ø;

  TYPE
    bereich       = Ø..seitenlänge;
    feldtyp       = (leer, markiert, dame);
    spielbrettyp = ARRAY[bereich, bereich] OF feldtyp;
    stellungstyp = RECORD x,y : bereich END;
    spielertyp    = (mensch, computer);

  VAR
    brett         : spielbrettyp;
    position      : stellungstyp;
    amzug         : spielertyp; (* kennzeichnet den am Zug
                                   befindlichen Spieler *)

  PROCEDURE beschreibung;
    (* gibt die spielanleitung und benutzerhinweise *)
    BEGIN (* wird hier nicht ausgeführt *) END;


  PROCEDURE startangaben;
    (* legt anziehenden spieler und startposition fest *)
    BEGIN (* wird hier nicht ausgeführt *) END;


  PROCEDURE partie;
    (* das eigentliche spiel, die zugfolge *)
    BEGIN (* wird weiter unten ausgeführt *) END;


  PROCEDURE gewinnentscheid;
    (* ermittelt den gewinner der partie *)
    BEGIN (* wird weiter unten ausgeführt *) END;


  BEGIN (* Hauptprogramm *)
    beschreibung;
    startangaben;
    partie;
    gewinnentscheid
  END.
```

In der Prozedur 'Startangaben' wird der anziehende Spieler bestimmt, und es wird die Anfangsposition der Dame durch jenen festgelegt. Wegen der Platzbeschränkung in diesem Band muß ich auf den Abdruck der Prozedur verzichten; interessierten Lesern kann ich einen Ausdruck des vollständigen Pascal-(oder auch BASIC-)Programms zusenden. Die Prozedur 'Gewinnentscheid' ist ganz einfach:

```
PROCEDURE gewinnentscheid;
  (* ermittelt den gewinner der partie *)

BEGIN
  IF amzug = mensch THEN
    writeln('Sie haben verloren.')
  ELSE
    writeln('Sie haben gewonnen!')
END; (* of gewinnentscheid *)
```

Interessant ist allein die Prozedur 'Partie'; ihre Grobstruktur lautet wie folgt:

```
PROCEDURE partie;
  (* das eigentliche spiel, die zugfolge *)

  VAR
    links, unten : bereich;

  PROCEDURE spielstandsausgabe;
    (* gibt den aktuellen spielstand aus *)
    BEGIN ( * wird im folgenden nicht ausgeführt *) END;

  PROCEDURE neuer-spielstand;
    (* der neue spielstand wird berechnet *)
    BEGIN (* im folgenden nicht ausgeführt, da sehr ein-
                                              fach *) END;

  PROCEDURE der-mensch-zieht;
    (* der spielzug des menschen wird eingelesen und aus-
                                              geführt *)
    BEGIN (* wird im folgenden dargestellt *) END;

  PROCEDURE der-computer-zieht;
    (* der spielzug des computers wird ermittelt und aus-
                                              geführt *)
    BEGIN (*  wird im folgenden genauer dargestellt *) END;

  FUNCTION partie-ende : boolean;
    BEGIN
      partie-ende := (position.x = Ø) AND (position.y = Ø)
    END; (* of partieende *)
```

```
BEGIN (* partie *)
  REPEAT
    spielstandsausgabe;
    IF amzug = mensch THEN BEGIN
      der-mensch-zieht;
      amzug := computer END
    ELSE BEGIN
      der-computer-zieht;
      amzug := mensch
    END(*of else*)
  UNTIL partie-ende
END; (*of partie*)
```

Die Prozedur "der Mensch zieht" lautet wie folgt:

```
PROCEDURE der-mensch-zieht;
  (*  der spielzug des Menschen wird eingelesen und aus-
                                                 geführt *)
  FUNCTION in-ordnung : boolean;
    (* prüft, ob der spielzug legal ist *)
    BEGIN (* wird nicht näher ausgeführt *)  END;

  PROCEDURE zugeingabe;
    (* der zug des menschen wird eingelsen *)
    BEGIN (* nicht ausgeführt, da einfach *) END;

  BEGIN
    REPEAT zugeingabe UNTIL in-ordnung;
    neuer-spielstand;

  END; (* of der mensch zieht *)
```

Analog dazu die Prozedur "der Computer zieht":

```
PROCEDURE der-computer zieht;
  (* der spielzug des computers wird ermittelt und aus-
                                                geführt *)
  VAR
    x,y : bereich;

  PROCEDURE zugausgabe;
    (* der ermittelte spielzug wird ausgegeben *)
    BEGIN (* hier nicht ausgeführt, da klar *) END;

  PROCEDURE zugermittlung;
    (* der computer ermittelt seinen besten zug *)
    BEGIN (* wird unten näher beschrieben *) END;

  BEGIN

    zugermittlung;
    neuer-spielstand;
    zugausgabe

  END; (* of der computer zieht *)
```

Die Spielstrategie steckt in der Prozedur 'Zugermittlung':

```
PROCEDURE zugermittlung;
  (* der computer ermittelt seinen besten zug *)
  VAR
    i : integer;

  PROCEDURE waagerechtmarkieren"
    (* eine zeile wird markiert und ggf. die dame entdeckt *)
    BEGIN (* wird unten näher erläutert *) END;

  PROCEDURE senkrechtmarkieren;
    (* eine zeile wird markiert und ggf. die dame entdeckt *)
    BEGIN (* analog zu waagerechtmarkieren *) END;

  PROCEDURE diagonalmarkieren;
    (* eine diagonale wird markiert und ggf. die Dame ent-
                                                   deckt *)
    BEGIN (* analog zu senkrechtmarkieren *) END;

  BEGIN
    FOR x := Ø TO seitenlänge DO
      FOR y := Ø TO seitenlänge DO
        brett[x,y] := leer;
    brett[x,y] := dame;
    links := Ø; unten := Ø;
```

```
    FOR x := Ø TO seitenlänge DO
      FOR y := Ø TO seitenlänge DO BEGIN
        IF brett[x,y] = leer THEN BEGIN
          writeln(x,' ',y); (* verluststellung *)
          waagerechtmarkieren;
          senkrechtmarkieren;
          diagonalmarkieren
        END;
        IF (links = Ø) AND (unten = Ø) THEN
          links := 1 (* Verlegenheitszug *)
      END (* of for *)
  END; (* of zugermittlung *)
```

Es bleibt die Prozedur "waagerecht markieren", die die von einer Verluststellung ausgehende Zeile markiert; findet sie auf einem Feld die Dame, so zieht sie diese waagerecht auf die Verluststellung, von der sie ausging und springt aus der Prozedur 'Zugermittlung' heraus.

```
PROCEDURE waagerechtmarkieren;
  (* eine zeile wird markiert und ggf. die dame entdeckt *)
  BEGIN
    i := 1;
    WHILE x+i <= seitenlänge DO BEGIN
      IF brett[x+i,y] = dame THEN BEGIN
        links := i; unten := Ø;
        exit(zugermittlung)
      END; (* of then *)
      brett[x+i,y] := markiert;
      i := i+1
    END (* of while *)
  END; (* of waagerechtmarkieren *)
```

3 VERGLEICH BASIC-Pascal

Beim folgenden BASIC-Programm wurde versucht, genau die eben beschriebene Struktur (Gliederung in Hauptprogramm und Prozeduren) nachzuahmen. Trotzdem wird längst nicht der Grad an Lesbarkeit und Verständlichkeit wie beim Pascal-Programm erreicht; u.a. aus folgenden Gründen:

- Prozeduren lassen sich in BASIC nicht benennen und unter ihrem Namen aufrufen,

- Variablen haben in BASIC nur zwei signifikante Zeichen,
- die Definition von Datentypen, wie z.B. unser "feldtyp = (leer, besetzt, dame)" ist in BASIC nicht möglich.

Ein großer Vorteil von BASIC gegenüber Pascal soll indessen nicht verschwiegen, sondern muß gebührend herausgestellt werden: BASIC ermöglicht schon zwölfjährigen Schülern, komplexe Spielprogramme (wie z.B. das vorliegende) zu entwickeln; in Pascal würden sie dagegen bereits an den Deklarationen scheitern. Ihr Arbeitsstil ist intuitiv und experimentierend, durch Versuch und Irrtum tasten sie sich an das ihnen vorschwebende Ziel heran: dergleichen ist in Pascal nicht möglich.

```
1000 : PRINT CHR$(147)
1010 : PRINT "      TREIB DIE DAME IN DIE ECKE
1011 : PRINT "      -------------------------
1012 :
1020 REM NIM-AEHNLICHES STRATEGIESPIEL, DAS VON DEN ALTEN CHINESEN
1021 REM ERFUNDEN UND VON W.A.WYTHOFF (1907) NEU ENTDECKT WURDE
1022 :
1099 :
1100 REM *** HAUPTPROGRAMM ********************************************
1101 :
1200 : GOSUB 2000 : REM BESCHREIBUNG
1300 : GOSUB 3000 : REM STARTANGABEN
1400 : GOSUB 4000 : REM PARTIE
1500 : GOSUB 5000 : REM GEWINNENTSCHEID
1900 : END
1901 :
1990 REM ENDE DES HAUPTPROGRAMMS ************************************
1998 :
1999 :
2000 REM ### PROZEDUR 'BESCHREIBUNG' #################################
2001 :
2100 : REM HIER SOLLTEN DIE SPIELREGELN UND BENUTZERHINWEISE STEHEN
2101 :
2900 : RETURN
2901 :
2990 REM ENDE DER PROZEDUR 'BESCHREIBUNG' ###########################
2999 :
3000 REM ### PROZEDUR 'STARTANGABEN' #################################
3001 :
3050 : LET N = 20 : REM SEITENLAENGE DES BRETTS
3100 : DIM B(N,N) : REM SPIELBRETT
3190 :
3200 : PRINT "GEBEN SIE DIE ANFANGSPOSITION EIN!
3201 :
3205 : PRINT
3210 : INPUT "X-KOORDINATE "; X
3220 : INPUT "Y-KOORDINATE "; Y
3221 :
3230 : IF X > N  OR Y > N  THEN PRINT "ZU GROSS!": GOTO 3210
3240 : LET B(X,Y) = 5
3300 : LET SP$ = "COMPUTER"
3301 :
3900 : RETURN
3901 :
3990 REM ENDE DER PROZEDUR 'STARTANGABEN' ###########################
3999 :
```

```
4000 REM ### PROZEDUR 'PARTIE' ##########################################
4001 :
4100 : REM === PROZEDURRUMPF 'PARTIE' ================================
4101 :
4110 :      GOSUB 4200 : REM SPIELSTANDSAUSGABE
4120 :      IF SP$ = "COMPUTER" THEN 4150
4121 :
4130 :   REM --- DER MENSCH IST AM ZUG
4131 :
4135 :      GOSUB 4300 : REM ZUG DES MENSCHEN
4140 :      LET SP$ = "COMPUTER"
4145 :      GOTO 4170  : REM PARTIE ZU ENDE?
4149 :
4150 :   REM --- DER COMPUTER IST AM ZUG
4151 :
4153 :      PRINT : PRINT "LASSEN SIE MICH UEBERLEGEN ...
4155 :      GOSUB 4400 : REM ZUG DES COMPUTERS
4160 :      LET SP$ = "MENSCH"
4165 :
4170 :   REM --- PRUEFUNG AUF PARTIEENDE
4171 :
4175 :      GOSUB 4900
4180 :      IF E$ = "WEITER" THEN 4110
4185 :
4190 :      RETURN
4191 :
4195 : REM ENDE DES PROZEDURRUMPFS 'PARTIE' ========================
4199 :
4200 : REM +++ UNTERPROZEDUR 'SPIELSTANDSAUSGABE' ++++++++++++++++++++
4201 :
4210 :   PRINT
4220 :   PRINT "SPIELSTAND: "; X; Y
4221 :
4290 :   RETURN
4291 :
4295 : REM ENDE DER UNTERPROZEDUR 'SPIELSTANDSAUSGABE' +++++++++++++++
4299 :
4300 : REM +++ UNTERPROZEDUR 'ZUG DES MENSCHEN' +++++++++++++++++++++
4301 :
4310 :   PRINT
4320 :   PRINT "SIE SIND DRAN!
4330 :   INPUT "WIEVIEL NACH LINKS "; L
4340 :   INPUT "WIEVIEL NACH UNTEN "; U
4350 :   IF L*U <> 0 AND L <> U THEN PRINT "FALSCHER ZUG!": GOTO 4330
4360 :   IF L = 0 AND U = 0 THEN PRINT "SIE MUESSEN ZIEHEN!": GOTO 4330
4361 :
4370 :   LET X = X-L : LET Y = Y-U
4380 :   IF X < 0 OR Y < 0 THEN 4330
4385 :   LET B(X,Y) = 5
4389 :
4390 :   RETURN
4391 :
4395 : REM ENDE DER UNTERPROZEDUR 'ZUG DES MENSCHEN' ++++++++++++++++
4399 :


4400 : REM +++ UNTERPROZEDUR 'COMPUTERZUG' +++++++++++++++++++++++++++
4401 :
4405 :   FOR R = 0 TO N : FOR S = 0 TO N : LET B(R,S) = 0 : NEXT S,R
4406 :   LET B(X,Y) = 5
4407 :   LET L = 0 : LET U = 0
4409 :
4410 :   FOR R = 0 TO N
4420 :     FOR S = 0 TO N
4430 :       IF B(R,S) = 1 THEN 4600
4435 :       PRINT R,S : REM AUSGABE DER VERLUSTSTELLUNG
4440 :
4450 :       LET I = 0
```

```
4460 :        LET I = I+1 : IF R+I > N THEN 4500
4470 :        IF B(R+I,S) <> 5 THEN LET B(R+I,S) = 1 : GOTO 4460
4480 :        LET L = I : LET U = 0 : GOTO 4700
4490 :
4500 :        LET I = 0
4510 :        LET I = I+1 : IF S+I > N THEN 4550
4520 :        IF B(R,S+I) <> 5 THEN LET B(R,S+I) = 1 : GOTO 4510
4530 :        LET L = 0 : LET U = I : GOTO 4700
4540 :
4550 :        LET I = 0
4560 :        LET I = I+1 : IF R+I > N OR S+I > N THEN 4600
4570 :        IF B(R+I,S+I) <> 5 THEN LET B(R+I,S+I) = 1 : GOTO 4560
4580 :        LET L = I : LET U = I : GOTO 4700
4590 :
4600 :      NEXT S
4610 :    NEXT R
4699 :
4700 :    IF L = 0 AND U = 0 THEN LET L = 1 : REM VERLEGENHEITSZUG
4710 :    PRINT "ICH ZIEHE"; L;"NACH LINKS UND"; U;"NACH UNTEN.
4720 :    LET X = X-L : LET Y = Y-U
4730 :
4790 :    RETURN
4791 :
4795 : REM ENDE DER UNTERPROZEDUR 'COMPUTERZUG' +++++++++++++++++++++
4799 :
4900 : REM +++ UNTERPROZEDUR 'PARTIEENDE' ++++++++++++++++++++++++++++
4901 :
4910 :    LET E$ = "WEITER"
4920 :    IF X = 0 AND Y = 0 THEN LET E$ = "ENDE"
4930 :
4990 :    RETURN
4991 :
4995 : REM ENDE DER UNTERPROZEDUR 'PARTIEENDE' ++++++++++++++++++++++
4999 :
5000 REM ### PROZEDUR 'GEWINNENTSCHEID' ##############################
5001 :
5100 : IF SP$ = "MENSCH"   THEN PRINT "SIE HABEN VERLOREN.
5200 : IF SP$ = "COMPUTER" THEN PRINT "SIE HABEN GEWONNEN!
5300 :
5900 : RETURN
5901 :
5990 REM ENDE DER PROZEDUR 'GEWINNENTSCHEID' #########################
5999 :
```

4 MATHEMATISCHE ENTDECKUNGEN

Beim Spiel gegen den Computer produziert dieser auf seiner Suche nach einem optimalen Zug die Folge der Zahlenpaare x(n), y(n), welche die Verluststellungen kennzeichnen:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x(n)	1	3	4	6	8	9	11	12	14	16	17	19	21	22	24	25
y(n)	2	5	7	10	13	15	18	20	23	26	28	31	34	36	39	41

An ihr lassen sich interessante mathematische Entdeckungen machen. Wir stellen zunächst fest:

(1) $$x(n) + y(n) = n$$

(2) x(n) ist die kleinste der bisher noch nicht aufgetretenen Zahlen.

Hieraus läßt sich die Folge der Verluststellungen ohne die aufwendige Suche in der Prozedur 'Zugermittlung' rein numerisch (rekursiv) gewinnen (und zu Beginn eines Spiels speichern). Dies ist wieder ein Beispiel für den Sachverhalt:

- Mit etwas Mathematik kann man sich aufwendiges Programmieren ersparen.

Sein Gegenstück lautet:

- Mit etwas Programmieren kann man (eventuell schwierige) Mathematik umgehen;

beide zusammen kennzeichnen das ambivalente Verhältnis von Computer und Mathematik, welches zunehmend den Unterricht zu beherrschen beginnt. -
Dem Kenner fällt sofort auf, daß in obiger Tabelle die Fibonacci-Folge versteckt ist; damit ist der Weg zum goldenen Schnitt nicht weit (vgl. [4]). Die Folgen x(n), y(n) sind in der Form

(3) $$x(n) = \mathrm{int}(n \cdot \varphi)$$

(4) $$y(n) = \mathrm{int}(n \cdot \varphi^2) \quad \text{mit} \quad \varphi = \frac{1 + \sqrt{5}}{2}$$

explizit darstellbar (die Programmieraufgabe ist jetzt schon fast trivial!). Hieraus läßt sich eine direkte Kennzeichnung der Verluststellungen entwickeln, die an die berühmte Dualdarstellung der Verluststellungen im Nim-Spiel erinnert.
Aus Platzgründen kann ich diese interessanten - etwa für eine Mathematik- oder Informatikarbeitsgemeinschaft bzw. für den sogenannten problemorientierten Unterricht (vgl. [8]) geeigneten - Möglichkeiten für mathematische Entdeckungen nicht näher eingehen; genaueres findet der Leser in [6], und weitere Literatur in [3, S. 80].

LITERATURVERZEICHNIS

[1] A v e r b a c h, B./C h e i n, O.: Mathematics. Problem Solving through Recreational Mathematics. San Francisco (Freeman) 1980

[2] B a u m a n n, R.: Computerspiele und Knobeleien - programmiert in Basic. Würzburg (Vogel) 1982

[3] B e r l e k a m p, E.R./C o n w a y, J.H./G u y, R.A.: Winning Ways for your mathematical plays. London (Academic Press) 1982

[4] C o x e t e r, H.S.M.: The golden section, phyllotaxis, and Wythoff's game. In: Scripta mathematica 19 (1953), 139

[5] G a r d n e r, M.: Mathematischer Karneval. Frankfurt - Berlin (Ullstein) 1975

[6] --- : Mathematical games: Cornering a Queen leads unexpectedly into corners of the theory of numbers. In: Scientific American, März 1977, 134-140

[7] G r u b e, K.H.: Zum Solitairespiel. In: Der Mathematikunterricht 26/2 (1980), 37-63

[8] K i e s s w e t t e r, K.: Spielen und Mathematiklernen. In: Zentralblatt für Didaktik der Mathematik 11/3 (1979), 109-112

[9] K r a w c z y k, R.: Endliche Automaten und Spiele. In: Der mathematische und naturwissenschaftliche Unterricht 31 (1978), 136-143

[10] M a r k w a l d, W.: Legespiele, eine Propädeutik für Algorithmen. In: Der Mathematikunterricht 18/4 (1972), 73-87

[11] M ü l l e r, G./ W i t t m a n n, E.: Der Mathematikunterricht in der Primarstufe. Braunschweig (Vieweg) 1977

[12] S c h r a g e, G.: Ein topologisches Spiel zum Sperner-Lemma. In: Praxis der Mathematik 14 (1972), 195-197

[13] --- : Strategiespiele und Gewinnstrategien. In: Mathematische Semesterberichte 28 (1981), 92-103

[14] --- : Die algorithmische Struktur strategischer Spiele. Erscheint demnächst

[15] --- : Rekursive Behandlung strategischer Spiele. Erscheint demnächst

[16] Schupp, H.: Funktionen des Spiels im Mathematikunterricht der Sekundarstufe I. In: Praxis der Mathematik 20 (1978), 107-112

[17] Trampler, S.: Natur, Umwelt und Spiele im Dienst problemorientierten Unterrichts. In: Zentralblatt für Didaktik der Mathematik 11/3 (1979), 103-104

[18] Wynands, A.: Spiele auf Graphen. In: Der Mathematikunterricht 19/2 (1973), 36-56

Zwei Knobeleien

von Rüdeger Baumann

Die folgenden Beispiele sollen zeigen,

- daß Pascal wohlstrukturiertes Programmieren unterstützt, während BASIC zum sogenannten Spaghetti-Code verführt,
- daß Pascal zur Modellierung realer Situationen geeignete Datenstrukturen besitzt, während BASIC hieran arm ist.

1 n-DAMEN-PROBLEME

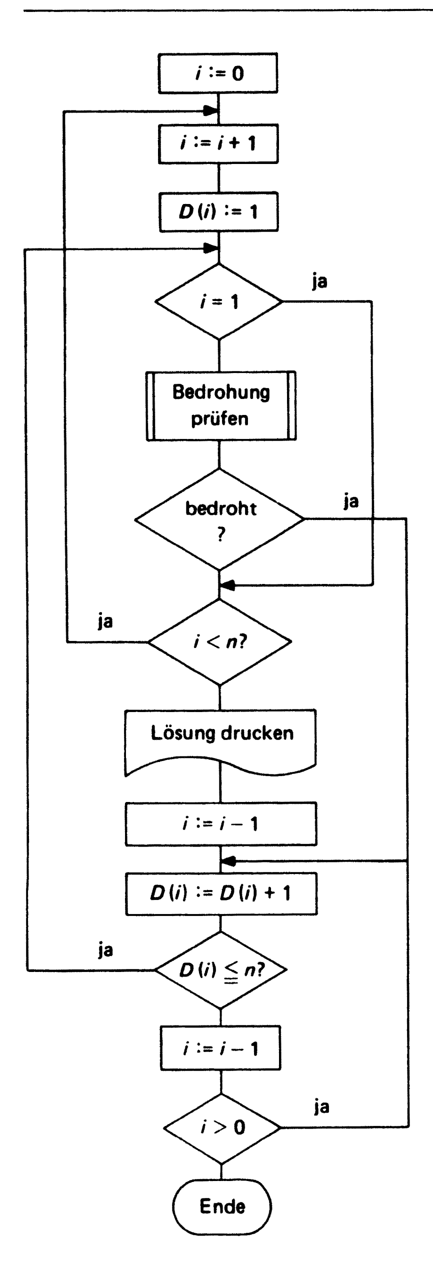
Ein klassisches Problem der Unterhaltungsmathematik lautet:

> Auf einem Schachbrett sollen acht Damen so plaziert werden, daß sie sich gegenseitig nicht schlagen können.

Selbst der große C.F. Gauß hat sich mit dieser Aufgabe beschäftigt; und natürlich interessierte er sich nicht nur für eine, sondern für alle Lösungen, mehr noch: er suchte ein systematisches Verfahren zur Erzeugung aller Lösungen. Es läßt sich wie folgt beschreiben:

"Man setzt von links nach rechts fortschreitend in jede Spalte des Schachbretts eine Dame, und zwar jedesmal an die tiefstmögliche Stelle. Wenn nun der Augenblick kommt, wo man keine Dame in einer Spalte aufstellen kann, so erhöht man den Platz der Dame in der vorhergehenden Spalte so lange, bis man fortfahren kann."

Die Stellung der Damen auf dem Schachbrett kennzeichnen wir durch Zahlen D(i); dabei ist i die Nummer der jeweiligen Spalte, in der die Dame steht (i = 1,2,...,n). Ein Flußdiagramm zum Gaußschen Verfahren ist in Bild 1 wiedergegeben.



B i l d 1: Flußdiagramm zum n-Damen-Problem

Das zugehörige BASIC-Programm lautet:

```
100   INPUT „Wieviele Damen"; n
110   DIM D(n)
120   LET i = 0
130   LET i = i + 1
140   LET D(i) = 1 : REM erste Dame aufs Brett
150   IF i = 1 THEN 190
160      FOR k = 1 TO i   1 : REM Bedrohung prufen
170         IF D(i) = D(k) OR ABS(D(i)   D(k)) = i   k THEN 230
180      NEXT k
190      IF i < n THEN 130
200         FOR k = 1 TO n   PRINT D(k); : NEXT k
210         PRINT : REM Lösung gefunden und gedruckt
220         LET i = i   1 : REM Dame vom Brett nehmen
230         LET D(i) = D(i) + 1 : REM Dame ruckt vor
240         IF D(i) < = n THEN 150
250            LET i = i - 1 : REM Dame vom Brett
260            IF i > 0 THEN 230
```

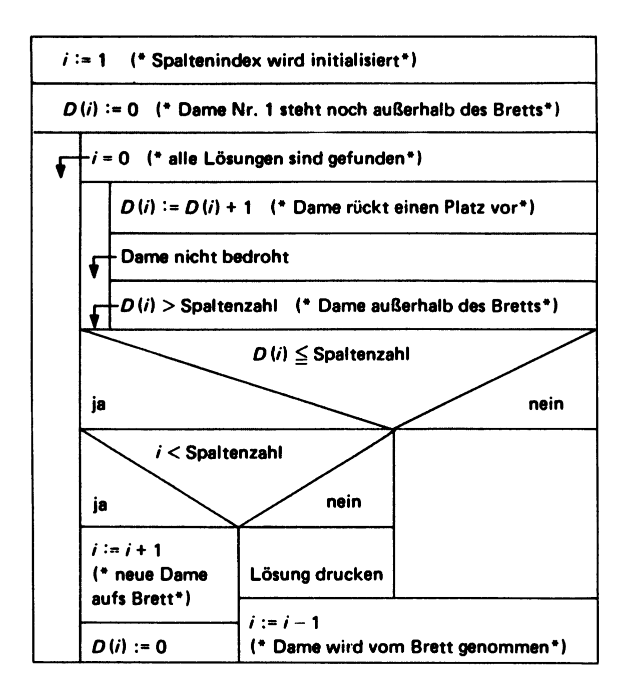
Zwei Damen in Spalte i und k bedrohen sich, wenn D(i) = D(k) ist (dann stehen sie in der gleichen Zeile) oder wenn $|D(i)-D(k)| = |i-k|$ (gleiche Diagonale): dies wird in Zeile 170 überprüft.

Das Programm ist erstaunlich kurz; dies verdankt es vor allem der in BASIC gegebenen Möglichkeit, an jede beliebige Stelle innerhalb eines Programms zu springen. Doch birgt sie Gefahren, denn das Springen (Anweisung GOTO), vor allem das Springen in Schleifen (wie es das Flußdiagramm anschaulich zeigt), macht Programme fehleranfällig und erschwert die Fehlersuche. Schlimmer noch: Programme dieser Art werden ab einem gewissen Umfang völlig unübersichtlich und ein Beweis der Korrektheit (hier: Beweis der Tatsache, daß mit dem Programm alle Lösungen des Acht-Damen-Problems gefunden werden) ist fast unmöglich.

Wollen wir das Gaußsche Verfahren in der Programmiersprache Pascal darstellen, müssen wir die 'Spaghetti-Fäden' zunächst entwirren, d.h. dem Algorithmus eine saubere Struktur geben.

Sie läßt sich am besten in einem Struktogramm ausdrücken (zu dem obigen Flußdiagramm dagegen gibt es kein Struktogramm!). Dies ist in B i l d 2 angegeben.

Als Preis für die Wohlstrukturiertheit müssen wir die doppelte Abfrage "D(i) > Spaltenzahl" bzw. "D(i) ≤ Spaltenzahl" zahlen; sie kam im BASIC-Programm nur einmal vor (Zeile 240). Die



B i l d 2: Struktogramm zum n-Damen-Problem

Blöcke "Dame nicht bedroht" und "Lösung drucken" legen wir in Unterprogramme (Prozeduren): dies fördert die Übersichtlichkeit.

Das Struktogramm läßt sich unmittelbar in Pascal übersetzen:

```
PROGRAM achtdamen;
   CONST
      spaltenzahl = 8;
   VAR
      brett : ARRAY [1.. spaltenzahl] OF integer;
      index: integer; (* Spaltenindex der Damen *)
   FUNCTION bedroht (i: integer) : boolean;
      VAR
         k : integer;
      BEGIN
         bedroht := false;
         FOR k := 1 TO i - 1 DO
            IF (brett [i] = brett [k]) OR (abs (brett [i]   brett [k]) = i   k)
               THEN BEGIN
                  bedroht := true; exit (bedroht)
               END (* of then *)
      END; (* of bedroht *)
   PROCEDURE ausgabe;
      VAR
         k : integer;
      BEGIN
         FOR k := 1 TO spaltenzahl DO write (brett [k] : 4);
         writeln
      END; (* of ausgabe *)
   BEGIN (* Hauptprogramm *)
      index .= 1; brett [index] := 0;
      WHILE index > 0 DO BEGIN
         REPEAT
            brett [index] := brett [index] + 1  .
         UNTIL NOT bedroht (index) OR (brett [index] > spaltenzahl);
         IF brett [index] < = spaltenzahl
            THEN
               IF index < spaltenzahl
                  THEN BEGIN
                     index  = index + 1; (* neue Dame *)
                     brett [index] := 0 END (* of then *)
                  ELSE BEGIN
                     ausgabe; (* Lösung gefunden *)
                     index := index   1 (* Dame vom Brett *)
                  END (* of else *)
               ELSE index := index   1 (* Dame vom Brett *)
            END (* of while *)
      END
```

Mit diesen Programm lassen sich die 92 Lösungen des Acht-Damen-Problems, angefangen bei 15863724, bis 84136275 erzeugen; dabei läuft das Pascal-Programm auf einer Microengine etwa 100mal so schnell wie das BASIC-Programm auf einem CBM.

Beachtet man, daß sich aus einer Lösung durch Drehen und Spiegeln weitere erzeugen lassen, reicht es, den Computer nur bis D(1) = 3 rechnen zu lassen: dies ermöglicht eine drastische Verkürzung der Rechenzeit. Übrigens kann das Pascal-Programm auch rekursiv formuliert werden; eine Möglichkeit, die es in BASIC nicht gibt.

2 JOSEFS-SPIEL

Eine besondere Stärke von Pascal ist das reiche Angebot an höheren Datenstrukturen: es gibt - wie in BASIC - das Feld (array); darüber hinaus aber Menge (set), Verbund (record), sequentielle Datei (file) sowie den Zeigertyp (pointer). Letzterer ist besonders interessant, weil er die Möglichkeit bietet, Variablen dynamisch zu kreieren. Als Exempel hierfür dienen das sogenannte Josefs-Spiel [1], ebenfalls ein altes Problem der Unterhaltungsmathematik, welches in allgemeiner Fassung wie folgt lautet:

> n Personen stehen im Kreis, jeder k-te wird ausgeschieden, wobei sich der Kreis sofort wieder schließt. Gesucht ist die Reihenfolge der Ausgeschiedenen.

In BASIC lösen wir es so: Wir denken uns die Kriegerschar (in der alten Geschichte, die dem Problem zugrundeliegt, handelt es sich um Soldaten) als verkettete Liste, indem wir in ein Feld mit n Plätzen die Nummer des jeweils rechts stehenden Kriegers, also des Nachfolgers in der Zählrichtung, schreiben:

k(1)	k(2)	k(3)	...	k(n-1)	k(n)
2	3	4	...	n	1

Das heißt z.B.: Krieger 3 ist Nachfolger von Krieger 2, und Krieger 1 ist Nachfolger von Krieger n: der Ring ist geschlossen.

Das Ausscheiden eines Kriegers wird nun durch die Anweisung k(z) := k(k(z)) realisiert: auf Platz z des Feldes steht jetzt nicht mehr der alte Nachfolger k(z), sondern dessen Nachfolger k(k(z)), d.h. k(z) wurde ausgeschieden. Er wird bei jedem künftigen Durchlaufen des Feldes übersprungen. Das Ausscheiden wird so lange fortgeführt, bis nur noch ein Krieger übrig ist: er ist dann sein eigener Nachfolger. Die Abbruchbedingung lautet also k(z) = z. Damit läßt sich das BASIC-Programm unmittelbar hinschreiben:

```
100   PRINT"           Josefs-Spiel"
101   PRINT"           ................"
102   :
110  REM Demonstriert verkettete Listen
111   :
120  REM  * Eingabe *
121   :
130   INPUT „Wieviel Krieger"; n
140   INPUT „Der wievielte soll jeweils ausgezählt werden"; s
190   :
200  REM * Aufstellen im Kreis *
201   :
210   DIM k(n)
220   FOR z = 1 TO n   1 : LET k(z) = z + 1 : NEXT z
230   LET k(n) = 1 : REM Schließen des Rings
290   :
300  REM * Auszählen *
301   :
310   LET z = n : REM Rückstellen des Zeigers z
320   FOR i = 1 TO s - 1 : LET z = k(z) : NEXT i
330   PRINT k(z);
340   LET k(z) = k(k(z)) : REM Ausscheiden der Nr. z
350   :
360   IF k(z) < > z THEN 320 : REM Noch nicht alle ausgezählt
370      PRINT z : REM Dies ist der letzte
390   END
```

Dialog:

```
Wieviel Krieger? 40
Der wievielte soll jeweils ausgezahlt werden? 7
 7  14  21  28  35   2  10  18  26  34   3  12
22  31  40  11  23  33   5  17  30   4  19  36
 9  27   6  25   8  32  16   1  38  37  39  15
29  13  20  24
```

In Pascal könnte man dieses Programm natürlich mittels des Datentyps ARRAY nachvollziehen; eine wesentlich elegantere Lösung bieten aber Variablen vom Zeigertyp. (Aus Platzgründen ist es hier leider nicht möglich, den Begriff des Zeigertyps ausführlich zu entwickeln; man vergleiche dazu [2] oder [1].)

Das zugehörige Pascal-Programm ist auf der nächsten Seite abgedruckt.

Dieses Programm hat vor dem BASIC-Programm den Vorteil, daß die Anzahl der aufzustellenden Personen vorher nicht bekannt zu sein braucht. Weiterhin können wir die Namen der Personen (nicht nur ihre Nummern) eingeben: dies wird durch die Benutzung des Datentyps Verbund (RECORD) ermöglicht. Er ist besonders für die Dateibearbeitung wichtig, existiert aber in BASIC leider nicht.

Pascal-Programm "Josephsspiel":

```
PROGRAM josefsspiel;
  TYPE
    zeiger = ↑ soldat;
    soldat = RECORD
               name       : string;
               nachfolger : zeiger
             END; (* of soldat *)
  VAR
    erster, krieger : zeiger;
    schrittweite    : integer;
  PROCEDURE aufstellenimkreis;
    PROCEDURE erzeuge (VAR k : zeiger);
      BEGIN
        new (k);
        writeln ('Name?');
        read (k ↑ name)
      END; (* of erzeuge *)
    BEGIN (* of aufstellenimkreis *)
      erzeuge (erster);
      krieger := erster;
      REPEAT
        erzeuge (krieger ↑ . nachfolger);
        krieger := krieger ↑ . nachfolger
      UNTIL eof;
      krieger ↑ . nachfolger := erster
    END; (* of aufstellenimkreis *)
  PROCEDURE auszahlen (schritt : integer);
    VAR
      i : integer;
    BEGIN
      REPEAT
        FOR i := 1 TO schritt   1 DO
          krieger := krieger ↑ . nachfolger;
        write (krieger ↑ . nachfolger ↑ . name : 6);
        krieger ↑ . nachfolger := krieger ↑ . nachfolger ↑ . nachfolger
      UNTIL krieger ↑ . nachfolger = krieger;
      write (krieger ↑ . name   6)
    END; (* of auszählen *)
  BEGIN (* Hauptprogramm *)
    writeln;
    writeln('              Josefs-Spiel');
    writeln('              ................');
    writeln; writeln;
    writeln('Geben Sie die Namen ein!');
    aufstellenimkreis;
    writeln('Der wievielte soll jeweils ausgezählt werden?');
    read(schrittweite);
    auszahlen(schrittweite)
  END.
```

LITERATURVERZEICHNIS

[1] B a u m a n n, R.: Programmieren mit PASCAL. Würzburg (Vogel) 1980

[2] S c h n e i d e r, W.: PASCAL. In: Taschenrechner + Mikrocomputer Jahrbuch 1981

10 Beispiele

von Dietmar Herrmann

1 QUERSUMME

Ganzzahlige Division durch 1Ø und Restbildung trennt die Einerziffer einer Dezimalzahl ab.
Z.B. gilt:

```
7249 mod 1Ø = 9
7249 div 1Ø = 724.
```

Setzt man das Verfahren mit dem Quotienten fort, so kann man schrittweise alle Ziffern abtrennen und die Quersumme berechnen.
Dieses Vorgehen kann direkt in Pascal programmiert werden:

```
quersumme := Ø;
repeat
     quersumme := quersumme + n mod 1Ø;
     n := n div 1Ø
until n = Ø;
```

Für die Funktion quersumme muß jedoch noch eine Variable quers eingeführt werden, da sich sonst die Funktion selbst aufruft und somit rekursiv berechnet wird.
Zur Ausführung der MOD- und DIV-Funktionen muß die Zahl n als ganzzahlig vereinbart werden. Dies schränkt die Größe der Zahl ein; bei dem hier benutzten Compiler gilt nämlich

```
MAXINT = 32767.
```

In BASIC können die Funktionen MOD und DIV mit Hilfe der INT-Funktion dargestellt werden:

```
a mod b  entspricht a - int(a/b)*b
a div b  entspricht int(a/b).
```

Da man aber in BASIC mit Hilfe der MID$-Funktion alphanumerische Variablen in einzelne Zeichen verwandeln kann, liegt ein anderes Vorgehen nahe:

Die Zahl Z wird durch

```
Z$ = STR$(Z)
```

in eine Ziffernkette verwandelt. Die Quersumme kann dann wie folgt berechnet werden:

```
Q = Ø
FOR I=1 TO LEN(Z$)
Q = Q + VAL(MID$(Z$,I,1))
NEXT I
```

Die VAL-Funktion wandelt die Ziffern wieder in Zahlen um. Da die meisten BASIC-Interpreter alphanumerische Variablen bis zu 255 Zeichen zulassen, kann man damit auch die Quersumme sehr großer Zahlen berechnen.

```
100 program quersumme(input,output);
110 var    zahl : integer;
120 (*                                           *)
130 function quersumme(n:integer):integer;
140 var quers:integer;
150 begin
160         quers:=0;
170         repeat
180              quers:=quers+n mod 10;
190              n:=n div 10
200         until n=0;
210         quersumme:=quers
220 end;(* quersumme *)
230 (*                                           *)
240 begin
250         writeln('Berechnung der Quersumme');
260         writeln;
270         writeln('gib natuerliche Zahl ein!');
280         read(zahl);
290         writeln('die Quersumme von',zahl:6,'=',quersumme(zahl):3)
300 end.
```

```
100 REM QUERSUMME EINER ZAHL
110 PRINTCHR$(147)
120 :
130 INPUT"EINGABE DER ZAHL";Z
140 :
150 REM UMWANDLUNG IN EINE ALPHANUMERISCHE VARIABLE
160 Z$=STR$(Z)
170 :
180 Q=0
190 FOR I=1 TO LEN(Z$)
200 Q=Q+VAL(MID$(Z$,I,1))
210 NEXT I
220 :
230 PRINT"QUERSUMME VON";Z;"=";Q
240 END
READY.
QUERSUMME

EINGABE DER ZAHL? 123456789
QUERSUMME VON 123456789 =45
```

2 EWIGER KALENDER

Ein bekannter Algorithmus zur Berechnung des Wochentags für ein gegebenes Datum ist das Verfahren von Zeller.
Für den Tag T, den Monat M und das Jahr J ergibt sich der Wochentag W aus der Formel

$$W = [2.6M-0.2] + J + [J/4] + [H/4] - 2H + T$$

dabei ist H die Nummer des Jahres innerhalb des Jahrhunderts. Die eckigen Klammern deuten auf die Anwendung der INT-Funktion hin. Zu beachten ist, daß Monate und Jahre nach dem spätrömischen Kalender berechnet werden müssen:

Januar und Februar sind die Monate 11 und 12 des Vorjahres;
März ist der Monat 1 und so fort.

Der Wochentag ist natürlich nur mod 7 bestimmt; es bedeutet

Ø Sonntag
1 Montag
2 Dienstag
usw.

In Pascal können die Variablen Tag, Monat und Jahr zur Kontrolle der Eingabe als Unterbereichstypen definiert werden:

```
tag = 1..31;   monat = 1..12;   jahr = 1582..2Ø99.
```

Ein Jahr vor 1582 einzugeben ist nicht sinnvoll, da der Gregorianische Kalender damals noch nicht gültig war.
Die Zellersche Formel wird als Funktion Wochentag definiert. Mittels der Mehrfach-Alternativanweisung CASE .. OF wird der gesuchte Wochentag ausgedruckt.

In BASIC müssen die MOD- und DIV-Funktion durch die INT-Funktion simuliert werden, wie es beim Programm Quersumme gezeigt wurde. Zur Ausgabe des Wochentags wird am besten die Mehrfach-Sprunganweisung ON .. GOTO benützt. Damit keine Zeile mehrfach durchlaufen wird, muß jede Zeile mit einer weiteren Sprunganweisung enden. Am Fehlen der GOTO-Anweisungen zeigt sich die Strukturiertheit der Programmiersprache Pascal besonders gut.

```
100 program  ewiger_kalender(input,output);
110 (* dieses Programm berechnet zu jedem Datum
120 des gregor.Kalenders den Wochentag
130 nach dem Verfahren des Geistl.Zeller*)
140 type  jahr  = 1582..2099;
150       monat = 1..12;
160       tag   = 1..31;
170 var   jhr   : jahr;
180       mon   : monat;
190       tg    : tag;
200 (*                                             *)
210 function wochentag(t:tag;m:monat;j:jahr):integer;
220 var jr,jhdt:integer;
230 begin
240    if m>2 then m:=m-2
250           else begin
260                m:=m+10;
270                j:=j-1
280                end;
290    jr:=j mod 100;
300    jhdt:= j div 100;
310    wochentag:=((13*m-1)div 5+jr div 4+jhdt div 4+jr+t-2*jhdt) mod 7
320 end;
330 (*                                             *)
340 begin (*Hauptprogramm *)
350 writeln('Gib Tag,Monat und Jahr ein!');
360    read(tg,mon,jhr);
370    write('der ',tg:2,'.',mon:2,'.',jhr:4,' ist ein ');
380    case wochentag(tg,mon,jhr) of
390         0:writeln('Sonntag');
400         1:writeln('Montag');
410         2:writeln('Dienstag');
420         3:writeln('Mittwoch');
430         4:writeln('Donnerstag');
440         5:writeln('Freitag');
450         6:writeln('Samstag')
460         end (* case *)
470 end.
```

```
100 REM EWIGER KALENDER
110 PRINTCHR$(147)
120 :
130 REM NACH DEM VERFAHREN VON ZELLER WIRD FUER JEDES DATUM
140 REM DES GREGORIANISCHEN KALENDERS DER WOCHENTAG BERECHNET
150 :
160 PRINT"        EWIGER KALENDER "
170 PRINT
180 INPUT"TAG,MONAT,JAHR";T,M,J
190 IF J<1582 THEN PRINT"JAHR VIERSTELLIG EINGEBEN":GOTO 180
200 PRINT"DER";T;".";M;".";J;"IST EIN ";
210 :
220 REM ZELLERSCHE FORMEL
230 IF M>2 THEN M=M-2:GOTO 250
240 M=M+10:J=J-1
250 H=INT(J/100)
260 J=J-100*H
270 W=INT(J/4)+INT(H/4)+INT((13*M-1)/5)+T+J-2*H
280 IF W>0 THEN W=W-7*INT(W/7):GOTO 300
290 IF W<0 THEN W=W+7:GOTO 290
300 :
310 ON W+1 GOTO 320,330,340,350,360,370,380
320 PRINT"SONNTAG":GOTO 400
330 PRINT"MONTAG":GOTO 400
```

```
 340 PRINT"DIENSTAG":GOTO 400
 350 PRINT"MITTWOCH":GOTO 400
 360 PRINT"DONNERSTAG":GOTO 400
 370 PRINT"FREITAG":GOTO 400
 380 PRINT"SAMSTAG"
 390 :
 400 END
READY.

EWIGER KALENDER

DER 24 . 12 .1982 IST EIN FREITAG
```

3 SIMULATION DER TEILERFREMDHEIT

Das Nachvollziehen eines Zufallsexperiments mit Hilfe von Zufallszahlen nennt man Monte-Carlo-Simulation.
Im folgenden soll die Wahrscheinlichkeit, daß zwei natürliche Zahlen teilerfremd sind, simuliert werden.
Dazu werden mittels eines Zufallszahlen-Generators 1ØØØ Paare von dreistelligen Zufallszahlen bestimmt. Durch Anwendung des Euklidschen Algorithmus wird der größte gemeinsame Teiler (ggT) ermittelt. Ist der ggT der beiden Zufallszahlen 1, so wird der Zähler der absoluten Häufigkeit weitergezählt.
Die relative Häufigkeit stellt den Monte-Carlo-Schätzwert für die gesuchte Wahrscheinlichkeit dar. Man kann beweisen, daß hier die Wahrscheinlichkeit

$$\frac{6}{\pi^2} = 0.6079..$$

ist.
Da im Standard-Pascal kein Zufallszahlen-Generator definiert ist, muß ein solcher definiert werden. Dafür gibt es zahlreiche Möglichkeiten; hier wird die Folge

$$x_{n+1} = e^{\pi + x_n} - \text{TRUNC}(e^{\pi + x_n})$$

mit einer zufällig gewählten Startzahl x_0 benützt. Sie ist im folgenden Programm als Funktion RANDOM gegeben.

Der Euklidsche Algorithmus kann durch eine Wiederholungsanweisung formuliert werden:

```
repeat
      rest := m mod n;
      m := n; n := rest
until rest = Ø;
ggT := m;
```

In BASIC ist ein Zufallszahlen-Generator implementiert, der meist durch RND(X) aufgerufen wird. Dagegen gibt es keine Funktionsprozeduren. Der Euklidsche Algorithmus muß somit als Unterprogramm eingegeben werden.

```
100 program teilerfremdheit(output);
110 (*  Monte-Carlo-Simulation der Wahrscheinlichkeit
120 dass 2 natuerliche Zahlen teilerfremd sind *)
130 const  anzahl       = 1000;
140        tausend      = 1000;
150        pi           = 3.141592653;
160 var    zaehler,i,a,b:integer;
170        haeufigk,x    : real;
180 (*                                           *)
190 function random:real;
200 begin
210     x:=exp(x+pi);
220     x:=x-trunc(x);
230     random:=x
240 end;
250 (*                                           *)
260 function ggt(m,n:integer):integer;
270 var  rest:integer;
280 begin
290     repeat
300          rest:= m mod n;
310          m:=n;n:=rest
320     until rest=0;
330     ggt:=m
340 end;
350 (*                                           *)
360 begin (*Hauptprogramm *)
370    zaehler:=0; (* Setzen des Zaehlers*)
380    x:=7.5721983e-01; (* Start von Random *)
390    for i:=1 to anzahl do
400         begin
410         a:=trunc(tausend*random);
420         b:=trunc(tausend*random);
430         if b<>0 then if ggt(a,b)=1 then zaehler:=zaehler+1
440         end;
450    haeufigk:=zaehler/anzahl;
460    writeln('Die rel.Haeufigkeit =',haeufigk:5:3)
470 end.
```

```
100 REM MONTE-CARLO-SIMULATION DER WAHRSCHEINLICHKEIT
110 REM DASS ZWEI NATUERLICHE ZAHLEN TEILERFREMD SIND
120 PRINT"SIMULATION DER TEILERFREMDHEIT"
130 :
140 Z=0
150 REM ERZEUGEN DREISTELLIGER ZUFALLSZAHLEN
160 FOR I=1 TO 1000
170 A=INT(1000*RND(1))
180 B=INT(1000*RND(2))
190 GOSUB 280
200 IF GGT=1 THEN Z=Z+1
210 NEXT I
220 :
230 REM AUSGABE
240 H=Z/1000
250 PRINT"DIE REL.HAEUFIGKEIT=";H
260 END
270 :
280 REM UNTERPROGRAMM FUER GGT
290 REM EUKLIDSCHER ALGORITHMUS
300 R=A-INT(A/B)*B
310 IF R=0 THEN GGT=B:RETURN
320 A=B:B=R
330 GOTO 300
READY.
```

```
SIMULATION DER TEILERFREMDHEIT

DIE REL.HAEUFIGKEIT= .613
```

4 CRAMERSCHE REGEL

Mit Hilfe der Cramerschen Regel lassen sich lineare Gleichungssysteme mit drei Unbekannten mittels Determinanten berechnen.

Beispiel:

$$\begin{aligned} 6x_1 - 4x_2 + 3x_3 &= 7 \\ 3x_1 + 5x_2 - 8x_3 &= -11 \\ 2x_1 \qquad\quad + 4x_3 &= 14 \end{aligned}$$

Die Hauptdeterminante D_o

$$\begin{vmatrix} 6 & -4 & 3 \\ 3 & 5 & -8 \\ 2 & 0 & 4 \end{vmatrix} = 202$$

verschwindet nicht und zeigt so, daß das obige Gleichungssystem eindeutig lösbar ist. Setzt man die rechte Seite des Systems für einen der Spaltenvektoren ein, so erhält man drei weitere Determinanten, aus deren Wert sich die Unbekannten berechnen lassen:

$$x = \frac{1}{D_o} \begin{vmatrix} 7 & -4 & 3 \\ -11 & 5 & -8 \\ 14 & 6 & 4 \end{vmatrix} = \frac{2\emptyset 2}{2\emptyset 2} = 1$$

$$y = \frac{1}{D_o} \begin{vmatrix} 6 & 7 & 3 \\ 3 & -11 & -8 \\ 2 & 14 & 4 \end{vmatrix} = \frac{4\emptyset 4}{2\emptyset 2} = 2$$

$$z = \frac{1}{D_o} \begin{vmatrix} 6 & -4 & 7 \\ 3 & 5 & -11 \\ 2 & \emptyset & 4 \end{vmatrix} = \frac{6\emptyset 6}{2\emptyset 2} = 3$$

In Pascal läßt sich das Einsetzen der Spaltenvektoren durch Wahl eines passenden Variablentyps direkt programmieren:

```
type vektor = array [1..3] of real;
```

Auch zur Determinanten-Berechnung kann eine geeignete Funktion definiert werden. Damit das Feld der Koeffizienten nicht kopiert werden muß, wird das Feld - wie auch in den folgenden Programmen - als globale Variable behandelt.
Da BASIC keine Funktionen mehrerer Variablen kennt, muß die Determinanten-Berechnung in einem Unterprogramm vorgenommen werden. Dazu muß allerdings die Variablenübergabe explizit programmiert werden, da BASIC nur über globale Variablen verfügt.

```
100 program cramersche_regel(output);
110 const  n=3;  (* Zahl der Unbekannten*)
120 type   vektor=array[1..n] of real;
130 var    a,b,c,d         : vektor;
140        hauptdet,x,y,z: real;
150        index           : 1..n;
160 (*                                         *)
170 function determinante(r,s,t:vektor):real;
180 begin
190 determinante:=r[1]*s[2]*t[3]+r[2]*s[3]*t[1]+r[3]*s[1]*t[2]
200               -r[3]*s[2]*t[1]-r[2]*s[1]*t[3]-r[1]*s[3]*t[2]
210 end;(* determinante *)
220 (*                                         *)
230 procedure eingabe;
240 begin
250 a[1]:=6.0;b[1]:=-4.0;c[1]:=3.0;d[1]:=7.0;
260 a[2]:=3.0;b[2]:=5.0;c[2]:=-8.0;d[2]:=-11.0;
270 a[3]:=2.0;b[3]:=0.0;c[3]:=4.0;d[3]:=14.0
280 end;
290 (*                                         *)
300 begin (* Hauptprogramm *)
310 eingabe;
320 hauptdet:=determinante(a,b,c);
330 writeln('Hauptdeterminante=',hauptdet);
340 if abs(hauptdet)<1.0e-10
```

```
350        then writeln('Gleichungsmatrix singulaer')
360        else begin
370             x:=determinante(d,b,c)/hauptdet;
380             y:=determinante(a,d,c)/hauptdet;
390             z:=determinante(a,b,d)/hauptdet;
400             writeln('x=',x,' y=',y,' z=',z);
410             end
420 end.
```

```
100 REM CRAMERSCHE REGEL FUER 3 UNBEKANNTE
110 PRINTCHR$(147)
120 PRINT"CRAMERSCHE REGEL"
130 :
140 DIM A(3,4),D(3,4)
150 FOR I=1 TO 3
160 READ A(I,1),A(I,2),A(I,3),A(I,4)
170 NEXT I
180 :
190 REM HAUPTDETERMINANTE
200 FOR I=1 TO 3
210 : FOR J=1 TO 3
220 : : D(I,J)=A(I,J)
230 : NEXT J
240 NEXT I
250 GOSUB 540
260 D0=D
270 IF D0=0 THEN PRINT"GLEICHUNGSSYSTEM NICHT EINDEUTIG LOESBAR":END
280 :
290 REM X-DETERMINANTE
300 FOR I=1 TO 3:D(I,1)=A(I,4):NEXT I
310 GOSUB 540
320 DX=D
330 :
340 REM Y-DETERMINANTE
350 FOR I=1 TO 3
360 : D(I,1)=A(I,1):D(I,2)=A(I,4)
370 NEXT I
380 GOSUB 540
390 DY=D
400 :
410 REM Z-DETERMINANTE
420 FOR I=1 TO 3
430 : D(I,2)=A(I,2):D(I,3)=A(I,4)
440 NEXT I
450 GOSUB 540
460 DZ=D
470 :
480 REM AUSGABE
490 PRINT"LOESUNG"
500 PRINT"X=";DX/D0,"Y=";DY/D0,"Z=";DZ/D0
510 END
520 :
530 REM UNTERPROGRAMM FUER DETERMINANTENBERECHNUNG
540 D=D(1,1)*D(2,2)*D(3,3)+D(1,2)*D(2,3)*D(3,1)+D(1,3)*D(2,1)*D(3,2)
550 D=D-D(1,3)*D(2,2)*D(3,1)-D(1,2)*D(2,1)*D(3,3)-D(1,1)*D(2,3)*D(3,2)
560 RETURN
570 :
580 DATA 6,-4,3,7
590 DATA 3,5,-8,-11
600 DATA 2,0,4,14
READY.
```

```
CRAMERSCHE REGEL

LOESUNG
X=1     Y=2     Z=3
```

5 KOMPLEXES RECHNEN

Die für viele Anwendungen benötigten komplexen Zahlen lassen sich in Pascal bequem als Verbund (RECORD) vereinbaren:

```
type komplex = record  re, im : real  end;
```

dabei stellen re und im den Real- bzw. Imaginärteil der komplexen Zahl dar.

Die Addition und Subtraktion der komplexen Zahlen ist komponentenweise erklärt:

$$(a+bi) \pm (c+di) = (a\pm c) + (b\pm d)i$$

Die Multiplikation ist definiert durch

$$(a+bi)(c+di) = (ac-bd) + (ad+bc)i.$$

Die Division kann als Multiplikation mit dem Kehrwert

$$(c+di)^{-1} = \frac{1}{\sqrt{c^2+d^2}}(c-di)$$

durchgeführt werden, dabei stellt der Nenner den Betrag der komplexen Zahl dar.
Alle Rechenoperationen können in Pascal als Prozedur erklärt werden; die Ergebnisse werden als Variablen-Parameter ans Hauptprogramm übergeben. Die Betragsrechnung kann als Funktion definiert werden, da hier ein reeller Wert erhalten wird.

In BASIC gibt es den Datentyp RECORD nicht. Die komplexen Zahlen werden deshalb als reelles Zahlenpaar aufgefaßt. Wie in den vorausgegangenen Programmen könnte man die Rechen-Prozeduren wieder als Unterprogramme schreiben. Dies bringt hier aber keine Vereinfachung, da diese Programmteile nur einmal durchlaufen werden. Dagegen wird die mehrmals benötigte Ausgabe als Unterprogramm formuliert.

```
100 program komplexes_rechnen(output);
110 type komplex = record
120                  re,im:real end;
130 var      a,b,c: komplex;
140 procedure ausgabe(x:komplex);
150 begin
160 if x.im>0 then write(x.re,'+i*':3,x.im)
170           else write(x.re,'-i*':3,abs(x.im));
180 writeln
190 end;(* of ausgabe*)
200 procedure summe(x,y:komplex;var z:komplex);
210 begin
220    z.re:=x.re+y.re;
230    z.im:=x.im+y.im;
240 end;
242 procedure differenz(x,y:komplex;var z:komplex);
244 begin
246    z.re:=x.re-y.re;
247    z.im:=x.im-y.im
249 end;
250 procedure produkt(x,y:komplex;var z:komplex);
260 begin
270    z.re:=x.re*y.re-x.im*y.im;
280    z.im:=x.re*y.im+x.im*y.re;
290 end;
300 function betrag(x:komplex):real;
310 begin
320    betrag:=sqrt(sqr(x.re)+sqr(x.im))
330 end;
340 procedure inverses(x:komplex;var z:komplex);
350 begin
360    z.re:=x.re/sqr(betrag(x));
370    z.im:=-x.im/sqr(betrag(x))
380 end;
390 procedure eingabe;
400 begin
410    a.re:=7.0;a.im:=-2.0;
420    b.re:=5.0;b.im:=3.0
430 end;
440 begin (*Hauptprogramm *)
445    eingabe;
450    summe(a,b,c);
460    write('Summe=');ausgabe(c);
473    differenz(a,b,c);
475    write('Differenz=');ausgabe(c);
478    produkt(a,b,c);
480    write('Produkt=');ausgabe(c);
490    inverses(b,c);
500    produkt(a,c,c);
510    write('Quotient=');ausgabe(c)
520 end.
```

```
100 REM GRUNDRECHENARTEN FUER KOMPLEXE ZAHLEN
110 PRINTCHR$(147)
120 PRINT"      KOMPLEXES RECHNEN":PRINT
130 DIM RE(2),IM(2)
140 :
150 REM EINGABE
160 READ RE(1),IM(1),RE(2),IM(2)
170 :
180 REM ADDITION
190 RE=RE(1)+RE(2)
200 IM=IM(1)+IM(2)
210 PRINT "SUMME=";
220 GOSUB 450
230 :
240 REM SUBTRAKTION
250 RE=RE(1)-RE(2)
260 IM=IM(1)-IM(2)
270 PRINT "DIFFERENZ=";
280 GOSUB  450
290 :
300 REM MULTIPLIKATION
310 RE=RE(1)*RE(2)-IM(1)*IM(2)
320 IM=RE(1)*IM(2)+RE(2)*IM(1)
330 PRINT "PRODUKT=";
340 GOSUB 450
350 :
360 REM DIVISION
370 BETRAG=RE(2)↑2+IM(2)↑2
380 RE=RE(1)*RE(2)+IM(1)*IM(2)
390 IM=-RE(1)*IM(2)+RE(2)*IM(1)
400 RE=RE/BETRAG : IM=IM/BETRAG
410 PRINT "QUOTIENT=";
420 GOSUB 450
430 END
440 :
450 REM UNTERPROGRAMM ZUR AUSGABE
460 IF IM<0 THEN PRINT RE;"-";ABS(IM);"*I":GOTO 480
470 PRINT RE;"+";IM;"*I"
480 PRINT:RETURN
490 :
500 DATA 7,-2,5,3
READY.
```

```
KOMPLEXES  RECHNEN

SUMME=12 + 1*I

DIFFERENZ= 2 -5*I

PRODUKT= 41 +11*I

QUOTIENT=.852941176 -.911764706*I
```

6 PRIMZAHLSIEB DES ERATOSTHENES

Ein bekanntes Verfahren zur Bestimmung von Primzahlen ist das Primzahlsieb des Eratosthenes (ca. 280-200 v. Chr.).
In der Menge {2,3,4,5,.....,N} werden zunächst alle Vielfachen von 2 - jedoch die 2 selbst nicht - gestrichen, wie man sagt,

"ausgesiebt". Sodann werden alle Vielfachen der nächst größeren, im Sieb verbleibenden Primzahl entfernt - jedoch nicht die Primzahl selbst. Das Verfahren setzt sich in der angegebenen Weise fort, bis die Vielfachen einer Primzahl gestrichen werden sollen, die größer als $\sqrt{N}$ ist. Die im Sieb verbliebenen Zahlen sind genau die Primzahlen der gegebenen Menge.

In Pascal kann das Sieb als Mengenvariable vereinbart werden:

```
sieb : set of  2..n;
```

Folgende Mengenoperationen sind definiert:

```
A + B  Vereinigungsmenge
A * B  Schnittmenge
A - B  Restmenge,
```

ebenso folgende Booleschen Operatoren

```
IN   Prüfung auf Element-Eigenschaft
<=   Prüfung auf Teilmengen-Eigenschaft.
```

Die Abfrage, ob ein Element der Menge angehört, kann somit formuliert werden als:

```
for zahl := 2 to n do if zahl in sieb then ...
```

Entsprechend kann das Aussieben durch Bilden der Restmenge

```
sieb := sieb - [vielfach*zahl]
```

durchgeführt werden.

Genausowenig wie den Typ RECORD kennt BASIC den Datentyp Menge. Man kann sich aber wieder mit einem Feld A(I) behelfen; gehört die Zahl i zur Menge, so wird A(I) = 1 gesetzt, andernfalls zu Null.

Das Sieb wird gefüllt durch

```
FOR I=2 TO N : A(I)=1 : NEXT I.
```

Eine Zahl j wird ausgesiebt durch

```
A(J) = Ø.
```

Die Ausgabe der im Sieb verbleibenden geschieht durch

```
FOR I=2 TO N : IF A(I)=1 THEN PRINT I;: NEXT I.
```

Wie man sieht, ist die Programmierung mit Mengen wesentlich eleganter. Ein Nachteil des Mengentyps in Pascal ist jedoch, daß Mengen nicht ausgedruckt werden können. Dasselbe gilt auch für Unterbereichstypen (siehe Programm Ewiger Kalender) und Verbunde (siehe Programm Komplexes Rechnen).
Ein zweiter Nachteil des Mengentyps ist, daß die meisten Compiler die maximale Elementeanzahl stark einschränken.
Der hier verwendete Compiler läßt nur 127 Elemente zu.

```
100 program eratosthenes(output);
110 (* Primzahlsieb des Eratosthenes *)
120 const n                   = 125;
130 var   sieb                : set of 2..n;
140       i,zahl,vielfach     : integer;
150 (*                                  *)
160 procedure ausgabe;
170 var   k:integer;
180 begin
190       for k:=2 to n do
200            if k in sieb then write(k:4)
210 end;(* ausgabe *)
220 (*                                  *)
230 begin (* Hauptprogramm *)
240       sieb:= [2..n];
250       for zahl:=2 to trunc(sqrt(n)) do
260            begin
270            if zahl in sieb then
280                 begin (*aussieben*)
290                 vielfach:=2;
300                 while vielfach <= n div zahl do
310                      begin
320                      sieb:=sieb-[vielfach*zahl];
330                      vielfach:=vielfach+1
340                      end
350                 end
360            end;
370            ausgabe
380 end.
```

```
100 REM PRIMZAHLSIEB DES ERATOSTHENES
110 PRINTCHR$(147)
120 PRINT"SIEB DES ERATOSTHENES":PRINT
130 :
140 INPUT"OBERE GRENZE";N
150 DIM P(N)
160 :
170 REM FUELLEN DES SIEBS
180 FOR I=2 TO N
190 : P(I)=1
200 NEXT I
210 :
220 REM AUSSIEBEN
230 FOR I=2 TO SQR(N)
240 : IF P(I)=0 THEN 290
250 : J=I*I
260 : FOR K=J TO N STEP I
270 : : P(K)=0
```

```
 280 : NEXT K
 290 NEXT I
 300 :
 310 REM AUSGABE
 320 PRINT"PRIMZAHLEN BIS";N;"SIND"
 330 FOR I=2 TO N
 340 : IF P(I)=1 THEN PRINT I;
 350 NEXT I
 360 END
READY.
```

```
PRIMZAHLSIEB DES ERATOSTHENES

OBERE GRENZE ? 125

DIE PRIMZAHLEN BIS 125 SIND

2  3  5  7  11  13  17  19  23  29  31
37  41  43  47  53  59  61  67  71  73
79  83  89  97  101  103  107  109 113
```

7 BINÄRES SUCHEN

Als Beispiel eines rekursiven Verfahrens sei das binäre Suchen in einer geordneten Liste behandelt.
Von der gegebenen Liste mit bekannten Index-Grenzen wird zunächst der mittlere Index bestimmt. Vergleicht man das zugehörige mittlere mit dem gesuchten Element, so wird entweder das gesuchte gefunden oder aber diejenige Hälfte der Liste bestimmt, in der es sich befindet. Der Suchvorgang beginnt in dieser Hälfte von neuem.
Das Verfahren endet sicher, da die Anzahl der zu durchsuchenden Elemente jeweils halbiert wird. Spätestens beim letzten Element entscheidet es sich, ob die Suche erfolgreich war oder nicht.

In Pascal läßt sich dieses rekursive Verfahren direkt programmieren: Die Prozedur binärsuche ruft sich bei jedem Halbierungsschritt mit veränderten Grenzen selbst auf. Die Variablen-Parameter index und ergebnis übermitteln das Ergebnis der Binärsuche. Die Suche endet spätestens, wenn die zu durchsuchende Liste nur noch 1 Element enthält, da bei einem erneuten Prozeduraufruf der obere Listenindex kleiner als der untere wird.

Da es in BASIC keine rekursiven Prozeduren gibt, muß das binäre Suchen iterativ programmiert werden.
Bezeichnet I,J,K den Index des kleinsten, größten bzw. mittleren Listenelements, so wird I=J gesetzt, wenn das gesuchte Element sich in der linken Listenhälfte befindet, andernfalls K=J.
Das Programm endet spätestens, wenn die Liste ein Element enthält; d.h. wenn K-I=1 gilt.

```
100 program binaersuche(output);
110 const  n     = 10;
120 type   liste= array[0..n] of integer;
130 var    zahl,stelle: integer;
140        gefunden    : boolean;
150        l           : liste;
160 (*                                                 *)
170 procedure binaersuche(unten,oben,z:integer;var index:integer;
180                                              var ergebnis:boolean);
190 var  mitte:integer;
200 begin
210 mitte:=(oben+unten) div 2;
220 if oben < unten then begin
230                       ergebnis:=false;
240                       index:=0
250                       end
260                  else
270    if l[mitte]=z then
280                       begin
290                       ergebnis:=true;
300                       index:=mitte
310                       end
320                  else
330    if l[mitte]>z then
340                       binaersuche(unten,mitte-1,zahl,stelle,gefunden)
350                  else
360                       binaersuche(mitte+1,oben,zahl,stelle,gefunden)
370 end; (* binaersuche *)
380 (*                                                 *)
390 procedure eingabe;
400 begin
410 (*sortierte Liste *)
420 l[1]:=18;l[2]:=23;l[3]:=27;
430 l[4]:=32;l[5]:=35;l[6]:=44;
440 l[7]:=50;l[8]:=66;
450 l[9]:=81;l[10]:=99
460 end; (* eingabe *)
470 (*                                                 *)
480 begin (* Hauptprogramm *)
490 eingabe;
500 writeln('gib gesuchte Zahl ein !');
510 read(zahl);
520 binaersuche(1,n,zahl,stelle,gefunden);
530 if gefunden then
540                 writeln('Gesuchtes Element an',stelle:3,'.Stelle')
550             else
560                 writeln('Gesuchtes Element nicht gefunden')
570 end.
```

```
100 REM BINAERES SUCHEN
110 PRINTCHR$(147)
120 PRINT "    BINAERES SUCHEN":PRINT
130 :
140 REM EINLESEN
150 READ N:DIM A(N)
160 FOR I=1 TO N
170 READ A(I)
180 NEXT I
190 :
200 INPUT"GESUCHTES ELEMENT";XSUCH
210 PRINT
220 :
230 REM ANFANGSWERTE DER INDEX-GRENZEN
240 I=0:K=N+1
250 :
260 REM SUCHSCHRITT
270 J=INT((I+K)/2)
280 IF XSUCH=A(J) THEN PRINT"GESUCHTES ELEMENT AN";J;".STELLE":END
290 IF XSUCH > A(J) THEN I=J:GOTO 310
300 K=J
310 IF K-I=1 THEN PRINT"GESUCHTES ELEMENT NICHT GEFUNDEN":END
320 GOTO 270
330 :
340 END
360 REM DATEN
370 DATA 10
380 DATA 18,23,27,32,35,44,50,66,81,99
READY.
```

```
BINAERES SUCHEN

GESUCHTES ELEMENT? 66
GESUCHTES ELEMENT AN 8.STELLE
```

8 INTERVALLSCHACHTELUNG FÜR NULLSTELLEN

Eine wichtige Anwendung des binären Suchens ergibt sich bei der Nullstellenbestimmung durch Intervallhalbierung.
Nach Eingabe eines Intervalls wird durch Berechnung des Funktionswerts an der Intervallmitte entschieden, ob sich die gesuchte Nullstelle in der linken oder rechten Intervallhälfte befindet. Sodann wird das Intervall auf die entsprechende Hälfte reduziert. Setzt man das Verfahren in der angegebenen Weise fort, so wird die Nullstelle durch immer enger werdende Grenzen eingeschlossen. Um sicher zu gehen, daß die Funktion im eingegebenen Intervall eine Nullstelle besitzt, wird die Funktion auf Vorzeichenwechsel geprüft. Denn nach dem Zwischenwertsatz für stetige Funktionen hat jede stetige Funktion in einem Intervall, in dem sie das Vorzeichen ändert, mindestens

eine Nullstelle. Obwohl danach die Nullstelle sicher gefunden wird, könnte es sein, daß die eingegebene Genauigkeit nicht erreicht wird. In diesem Fall wird eine entsprechende Meldung ausgegeben.

Analog wie bei der Binärsuche wird das Intervallhalbierungsverfahren als Prozedur intervallhalb definiert; sie ist jedoch iterativ formuliert, da iterative Verfahren meist schneller als rekursive sind. Die Intervallhalbierung endet, wenn die gesuchte Genauigkeit erreicht worden ist oder mehr als 35 Iterationsschritte benötigt worden sind. Die Variablen-Parameter mitte und ergebnis übergeben die Ergebnisse der Nullstellensuche an das Hauptprogramm.
Das BASIC-Programm entspricht wegen der Nicht-Rekursivität dem Pascal-Programm; jedoch sind alle Variablen global.

```
100 program intervallhalbierung(input,output);
110 var      nullstelle,x,y               :real;
120          vorzeichenwechsel,gefunden:boolean;
130 (*                                    *)
140 function funktion(x:real):real;
150 begin
160      funktion:=sqr(x)-2.0
170 end;
180 (*                                    *)
190 procedure intervallhalb(function f:real;a,b:real;var mitte:real;
200                                            var ergebnis:boolean);
210 const epsilon= 1.0e-06;
220 var   zaehler: integer;
230 begin
240 zaehler:=0;
250       repeat
260             mitte:=(a+b)/2.0;
270             zaehler:=zaehler+1;
280             if f(a)*f(mitte)>0 then a:=mitte
290                                 else b:=mitte
300       until (abs(b-a)< epsilon)or(zaehler>35);
310       ergebnis:=(zaehler<=35);
320 end;
330 (*                                    *)
340 begin (*Hauptprogramm *)
350 writeln('Gib Intervallgrenzen ein!');
360 read(x,y);
370 vorzeichenwechsel:=(funktion(x)*funktion(y)<=0);
380 if vorzeichenwechsel then
390       begin
400       intervallhalb(funktion,x,y,nullstelle,gefunden);
410       if gefunden then writeln('Nullstelle=',nullstelle:7:6)
420                   else writeln('Genauigkeit nicht erreicht')
430       end
440                      else writeln('kein Vorzeichenwechsel');
450 end.
```

```
 100 REM INTERVALLHALBIERUNG FUER NULLSTELLEN
 110 PRINTCHR$(147)
 120 PRINT"INTERV.HALBIERUNG FUER NULLSTELLEN":PRINT
 130 :
 140 REM EINGABE DER FUNKTION
 150 DEF FNF(X)=X*X-2
 160 :
 170 INPUT"INTERVALLGRENZEN A<B";A,B
 180 EPS=1E-6:REM REL.FEHLER
 190 :
 200 IF FNF(A)*FNF(B) <=0 THEN 240
 210 PRINT"KEIN VORZEICHENWECHSEL":END
 220 :
 230 REM INTERVALLHALBIERUNG
 240 M=(A+B)/2:REM INTERVALLMITTE
 250 IF FNF(A)*FNF(M)<0 THEN B=M:GOTO 270
 260 A=M
 270 IF ABS(B-A) > EPS*ABS(M) THEN 240
 280 :
 290 PRINT:PRINT"NULLSTELLE=";(A+B)/2
 300 END
READY.
```

```
INTERV.HALBIERUNG FUER NULLSTELLEN

NULLSTELLE= 1.41421356
```

9 ACHT-DAMEN-PROBLEM

Ein Verfahren zur Lösung eines Problems, das durch Probieren vor sich geht und bei dem Schritte, die in eine "Sackgasse" führen, wieder rückgängig gemacht werden können, heißt Backtracking-Verfahren.
Ein solches Verfahren soll nun zur Lösung des bekannten Acht-Damen-Problems benützt werden: 8 Damen sollen so auf einem Schachbrett plaziert werden, daß sie sich nach den Regeln des Schachspiels nicht bedrohen.
Wegen der Vielzahl der Lösungen ist das Problem kaum von Hand zu bewältigen. Es gibt nämlich 92 Lösungen, von denen jedoch nur 12 nicht durch Spiegelung oder Drehung auseinander hervorgehen.

Im folgenden Pascal-Programm wurden die Variablen wie folgt codiert:

	Bedeutung
x [i] = j	Dame i steht in der Spalte j
a [j] wahr	keine Dame steht in Zeile j
b [k] wahr	keine Dame steht in der Hauptdiagonalen oder einer dazu parallelen
c [k] wahr	keine Dame steht in der Nebendiagonalen oder einer dazu parallelen

	1	2	3	4	5	6	7	8
1	O							
2					O			
3								O
4						O		
5			O					
6							O	
7		O						
8				O				

Wie man der Skizze entnimmt, sind die Hauptdiagonale und ihre Parallelen durch gleiche Zeilen- und Spaltensummen gekennzeichnet; entsprechend die Nebendiagonale and ihre Parallelen durch gleiche Differenz aus Spalten- und Zeilenzahl.
Setzen einer Dame kann so formuliert werden:

```
x[i] := j; a[j] := b[i+j] := c[i-j] := false.
```

Entsprechend bedeutet

```
a[j] := b[i+j] := c[i-j] := true
```

das Entfernen einer Dame vom Brett. Eine Dame kann gesetzt werden, wenn gilt

```
a[j] and b[i+j] and c[i-j] = true.
```

Die Prozedur versuche setzt durch Probieren Damen aufs Brett: Gelingt dies ohne Bedrohung durch eine andere, so ruft sie sich selbst auf und setzt die nächste, andernfalls nimmt sie die Dame vom Brett. Sind 8 Damen plaziert, so wird die entsprechende Stellung ausgedruckt.

In BASIC muß das Setzen wieder iterativ formuliert werden. Der Zeilenindex i gibt gleichzeitig die Anzahl der bereits gesetzten Damen an. Weiterzählen von i bedeutet Setzen einer weiteren Dame, entsprechend wird i vermindert, wenn eine Dame vom Brett genommen wird. Die Bedrohung einer Dame wird wie folgt geprüft:

```
FOR K=1 TO I-1
IF D(I) = D(K) OR ABS(D(I)-D(K)) = I-K THEN ...
NEXT K
```

Das Weiterrücken einer Dame wird durch

```
D(I) = D(I) + 1
```

erreicht. Sind 8 Damen plaziert, so erfolgt ebenfalls die Ausgabe der Damenstellungen in graphischer Form.

```
100 program achtdamenproblem(output);
110 var   i:integer;
120       a:array[1..8] of boolean;(*keine Dame in Zeile i*)
130       b:array[2..16] of boolean;(*keine Dame in Hauptdiag.i*)
140       c:array[-7..7] of boolean;(*keine Dame in Nebendiag.i*)
150       x:array[1..8] of integer;(*Position in der Spalte i*)
160 (*                                   *)
170 procedure ausgabe;
180 var   k:integer;
190 begin
200    for k:=1 to 8 do
210    write(x[k]:4);
220    writeln;
230 end;(* ausgabe*)
240 (*                                   *)
250 procedure versuche(i:integer);
260 var   j:integer;
270 begin
280 for j:=1 to 8 do
290   if a[j] and b[i+j] and c[i-j] then
300   begin
310     x[i]:=j;
320     a[j]:=false;b[i+j]:=false;
330     c[i-j]:=false;
340     if i<8 then versuche(i+1)
350             else ausgabe;
360     a[j]:=true;b[i+j]:=true;
370     c[i-j]:=true
380   end
390 end;(* versuche*)
400 (*                                   *)
410 begin (* Hauptprogramm *)
420     for i:=1 to 8 do a[i]:=true;
430     for i:=2 to 16 do b[i]:=true;
440     for i:=-7 to 7 do c[i]:=true;
450     versuche(1)
460 end.
```

```
100 REM 8-DAMEN-PROBLEM
110 PRINTCHR$(147)
120 DIM D(8)
130 :
140 I=0 :REM ZEILENINDEX
150 Z=0 : REM ZAEHLER FUER LOESUNG
160 :
170 I=I+1 : REM NEUE DAME AUFS BRETT
180 D(I)=1
190 IF I=1 THEN 230
200 FOR K=1 TO I-1 :REM SPALTENINDEX
210 IF D(I)=D(K) OR ABS(D(I)-D(K))=I-K THEN 360
220 NEXT K
230 IF I<8 THEN 170
240 :
250 PRINTCHR$(147):REM AUSGABE
260 Z=Z+1:PRINT Z;".LOESUNG":PRINT
270 FOR K=1 TO 8
280 FOR L=1 TO 8
290 IF D(K)=L THEN PRINT"● ";:GOTO 310
300 PRINT"▒ ";
310 NEXT L
320 PRINT:PRINT
330 NEXT K
340 :
350 I=I-1 : REM DAME VOM BRETT
360 D(I)=D(I)+1 :REM DAME RUECKT VOR
370 IF D(I)<=8 THEN 190
380 I=I-1 :REM DAME VOM BRETT
390 IF I>0 THEN 360
READY.
```

```
8-DAMEN-PROBLEM

1.LOESUNG

● ▒ ▒ ▒ ▒ ▒ ▒ ▒
▒ ▒ ▒ ▒ ● ▒ ▒ ▒
▒ ▒ ▒ ▒ ▒ ▒ ▒ ●
▒ ▒ ▒ ▒ ▒ ● ▒ ▒
▒ ▒ ● ▒ ▒ ▒ ▒ ▒
▒ ▒ ▒ ▒ ▒ ▒ ● ▒
▒ ● ▒ ▒ ▒ ▒ ▒ ▒
▒ ▒ ▒ ● ▒ ▒ ▒ ▒
```

10 LOGIK-AUFGABE

Durch Rechnen mit Wahrheitswerten können Logikaufgaben wie folgende gelöst werden:

3 Freunde A,B,C kommen unter folgenden Bedingungen zu einer Party:

(1) A kommt nicht, wenn B nicht kommt

(2) B und C kommen gemeinsam oder gar nicht

(3) A kommt, wenn C nicht kommt und umgekehrt

In Pascal können Wahrheitswerte (BOOLEAN) wie folgt verknüpft werden:

A oder B	durch	a or b
A und B		a and b
A impliziert B		a <= b
A äquivalent B		a = b
A xor B (ausschließ.Oder)		a <> b

Die obigen Bedingungen sind erfüllt, wenn

```
(not b <= not a) and (b = c) and (a <> c)
```

wahr ist. Die entsprechenden Wahrheitswerte werden aus der Menge aller Kombinationen ausgewählt, die durch 3 verschachtelte Zählwiederholungs-Anweisungen von der Art

```
for a := true downto false do
```

erzeugt werden.

In BASIC muß das Rechnen mit Wahrheitswerten durch arithmetische Ausdrücke simuliert werden. Folgende Codierung wurde gewählt

wahr	entspricht	+1
falsch		-1
nicht A		-A
A und B		sgn(A+B-1)
A oder B		sgn(A+B+1)
A impliziert B		wenn A<=B dann 1, sonst -1
A äquivalent B		sgn(A*B)
A xor B		-sgn(A*B)

Die Wahrheitswert-Verknüpfungen werden in getrennten Unterprogrammen durchgeführt. Der obengenannte Lösungsterm wird schrittweise berechnet, indem das jeweilig benötigte Unterprogramm angesprungen wird. Die dazu notwendige Parameterübergabe wird mit Hilfe der Variablen P und Q durchgeführt. Analog zum Pascal-Programm werden alle Kombinationen der Wahrheitswerte mittels dreier verschachtelter Schleifen

```
FOR A=1 TO -1 STEP -2  usw.
```

erzeugt. Diejenigen Kombinationen der Wahrheitswerte, die den Wert 1 ergeben, werden ausgedruckt. Hier ergibt sich

A = -1   B = 1   C = 1;

nur B und C kommen zur Party.

In einigen BASIC-Dialekten - wie z.B. bei Commodore - sind die Booleschen Operatoren AND, OR und NOT definiert.
So gilt z.B.

```
NOT(0) = -1
-1 AND 0 = 0
-1 OR 0 = -1.
```

Es läßt sich zeigen, daß diese Verknüpfungen der Zahlen 0 und -1 den Gesetzen der Aussagenlogik entsprechen, wenn man -1 als "wahr" und 0 als "falsch" auffaßt.

```
100 program logikaufgabe(output);
110 var    a,b,c,erfuellt:boolean;
120 (*
130      A kommt nicht,wenn B nicht kommt
140      B und C kommen gemeinsam oder gar nicht
150      A kommt,wenn C nicht kommt und umgekehrt
160                                                    *)
170 begin
180 for a:=true downto false do
190      for b:=true downto false do
200          for c:=true downto false do
210              begin
220              erfuellt:=(not b<=not a) and (b=c) and (a<>c);
230              if erfuellt then writeln(a,b,c)
240              end
250 end.
```

```
100 REM LOGIKAUFGABE
110 PRINTCHR$(147)
120 :
130 REM A KOMMT NICHT,WENN B NICHT KOMMT
140 REM B UND C KOMMEN GEMEINSAM ODER GAR NICHT
150 REM A KOMMT,WENN C NICHT KOMMT UND UMGEKEHRT
160 :
170 FOR A=1 TO -1 STEP -2
180 FOR B=1 TO -1 STEP -2
190 FOR C=1 TO -1 STEP -2
200 P=-B:Q=-A
210 GOSUB 370:REM IMPLIKATION
220 W1=W:P=B:Q=C
230 GOSUB 420:REM AEQUIVALENZ
240 P=W1:Q=W
250 GOSUB 500:REM UND
260 W2=W:P=A:Q=C
270 GOSUB 460:REM EXKLUSIV-ODER
280 P=W2:Q=W
290 GOSUB 500:REM UND
300 IF W=1 THEN PRINTA,B,C
310 NEXT C
320 NEXT B
330 NEXT A
340 END
350 :
360 REM IMPLIKATION
370 IF P<=Q THEN W=1:GOTO 390
380 W=-1
390 RETURN
400 :
410 REM AEQUIVALENZ
420 W=SGN(P*Q)
430 RETURN
440 :
450 REM EXKLUSIV-ODER
460 W=-SGN(P*Q)
470 RETURN
480 :
490 REM UND
500 W=SGN(P+Q-1)
510 RETURN
READY.
```

```
LOGIK-AUFGABE

-1    1    1
```

Berechnung von Determinanten

von Karl Achilles

1 AUFGABENSTELLUNG

Jeder quadratischen Matrix läßt sich eindeutig eine Zahl zuordnen, welche man als Determinante bezeichnet. Für eine (2,2)-Matrix läßt sich die Determinante auf einfache Weise berechnen. Die allgemeine Darstellung einer solchen Matrix ist

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad (1)$$

Für die Determinante $|A|$ gilt dann:

$$|A| = a_{11} \cdot a_{22} - a_{12} \cdot a_{21} \qquad (2)$$

Zur Berechnung einer n-reihigen Determinante ($n > 2$) gibt es verschiedene Methoden, insbesondere die Entwicklung der Determinante nach Zeilen bzw. Spalten. Für die numerische Berechnung mit Hilfe von Computern eignet sich jedoch eher ein Algorithmus, welcher auf der sogenannten Verdichtung beruht.

2 BESCHREIBUNG DES LÖSUNGSWEGES

Mittels Verdichtung wird eine Determinante n-ter Ordnung auf eine solche (n-1)-ter Ordnung reduziert. Durch sukzessive Reduktion ergibt sich schließlich eine Determinante 2. Ordnung, welche sich nach (2) berechnen läßt.
Die Methode der Verdichtung sei im folgenden erläutert.
Eine n-reihige Determinante resultiert aus einer (n,n)-Matrix und hat folgende allgemeine Form:

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdot\cdot & \cdots & a_{3n} \\ \vdots & \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdot\cdot & \cdots & a_{nn} \end{vmatrix} \qquad (3)$$

Voraussetzung ist, daß das Element a_{11} von Null verschieden ist. Falls aber $a_{11} = 0$, so wird irgendein von Null verschiedenes Element a_{1i} durch Vertauschen von Spalte 1 mit Spalte i an die erste Stelle gebracht. Dadurch ändert sich das Vorzeichen der Determinante, aber nicht ihr Betrag. Falls in irgendeiner Zeile (oder Spalte) alle Elemente gleich Null sind, so hat die Determinante den Wert Null!
Ist die oben genannte Bedingung erfüllt, so werden folgende Operationen vorgenommen:

- Die Zeilen 2 bis n werden mit a_{11} multipliziert. Dadurch ergibt sich der a_{11}^{n-1}-fache Wert der Determinante.
- Zeile 2 − a_{21} · Zeile 1
  Zeile 3 − a_{31} · Zeile 1
  ⋮ ⋮
  Zeile n − a_{n1} · Zeile 1

Man erhält folgendes:

$$a_{11}^{n-1} \cdot |A| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & A_{n2} & & A_{nn} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} A_{22} & \cdots & A_{2n} \\ \vdots & & \vdots \\ A_{n2} & \cdots & A_{nn} \end{vmatrix}$$

Daraus folgt schließlich:

$$|A| = a_{11}^{2-n} \cdot \begin{vmatrix} A_{22} & \cdots & A_{2n} \\ \vdots & & \vdots \\ A_{n2} & & A_{nn} \end{vmatrix} \tag{4}$$

Die Determinante, bestehend aus den A_{ik}, ist nur noch von (n−1)-ter Ordnung. Für die A_{ik} gilt:

$$\boxed{A_{ik} = a_{11} \cdot a_{ik} - a_{i1} \cdot a_{1k}} \quad (i,k \text{ von } 2 \text{ bis } n) \tag{5}$$

Auf diese Weise läßt sich eine Determinante n-ter Ordnung auf eine solche der Ordnung 2 zurückführen.

3 VERBALE NOTATION DES ALGORITHMUS

```
ANFANG
   DET:=1
   SOLANGE ZEILENZAHL > 2 TUE ANFANG
      FALLS A(1,1)=Ø, DANN ANFANG
         SUCHE GRÖSSTES ELEMENT DER 1. ZEILE (PIVOT)
         FALLS ALLE ELEMENTE DER 1. ZEILE NULL, DANN
            AUSGABE (DET = Ø)    ENDE

         VERTAUSCHE SPALTE DES PIVOTS MIT 1. SPALTE
         DET:= -DET
      ENDE

      FÜR ZEILE :=2 BIS ZEILENZAHL TUE ANFANG
         FÜR SPALTE :=2 BIS ZEILENZAHL TUE
            A(ZEILE,SPALTE):= A(1,1)*A(ZEILE,SPALTE) -
                              A(ZEILE,1)*A(1,SPALTE)
      ENDE
      DET:= DET*A(1,1)^(2-ZEILENZAHL)
      FÜR ZEILE:=2 BIS ZEILENZAHL TUE ANFANG
         FÜR SPALTE :=2 BIS ZEILENZAHL TUE
            A(ZEILE-1,SPALTE-1):= A(ZEILE,SPALTE)
      ENDE
         ZEILENZAHL:= ZEILENZAHL-1
      ENDE
      DET:= DET*(A(1,1)*A(2,2) - A(1,2)*A(2,1))
      AUSGABE DET
   ENDE
```

4 PROGRAMMLISTINGS (BASIC und Pascal)

BASIC-Programm Determinante

```
ÜLIST

1000  REM   * BERECHNUNG EINER N-
     REIHIGEN DETERMINANTE *
1010  REM  ----------------------

1020  REM  KARL ACHILLES    2.2.82

1030  REM  ----------------------

1040  HOME : INVERSE
1050  PRINT "DETERMINANTE": NORMAL
     : VTAB (5)
1060  INPUT "GEBEN SIE DIE ZEILEN
     ZAHL AN: ";N
1070  IF N < 2 THEN  PRINT "N MUS
     S GROESSER ALS 1 SEIN !": GOTO
     1060
1080  DIM N,MA(N,N),HILF(N,N)
1090  PRINT : PRINT "UND NUN DIE
     MATRIXELEMENTE (ZEILENWEISE)
     ": PRINT : PRINT
1100  GOSUB 1240: REM  EINGABE
1110  HOME : VTAB (10): HTAB (10)

1120  PRINT "BITTE WARTEN ..."
1130  GOSUB 1340: REM  BERECHNUNG

1140  HOME
1150  PRINT "DIE DETERMINANTE HAT
     DEN WERT "DET
1160  END : REM  ENDE HAUPTPROGRA
     MM
1170  REM  ======================

1180  REM
1190  REM  *** UNTERPROGRAMME ***

1200  REM  ----------------------

1210  REM
1220  REM
1230  REM  ----------------------
     -
1240  REM  EINGABE DER MATRIXELEM
     ENTE
1250  FOR I = 1 TO N
1260  FOR J = 1 TO N
1270  PRINT "A("I","J")= ": INPUT
     MA(I,J)
1280  NEXT J,I
1290  RETURN : REM  ENDE EINGABE
1300  REM  ----------------------

1310  REM
1320  REM
1330  REM  ----------------------
1340  REM  BERECHNUNG DER DETERMI
     NANTE
1350 HIDET = 1:ZZ = N
1360  IF ZZ = 2 THEN 1620: REM  B
     ERECHNE 2-REIHIGE DETERMINAN
     TE ELEMENTAR
1370  IF MA(1,1) <  > 0 THEN 1520

1380  REM  SUCHE PIVOTELEMENT (1.
     ZEILE)

1390  FOR K = 2 TO ZZ
1400  IF  ABS (MA(1,K)) >  ABS (M
     A(1,K - 1)) THEN PIVOT = MA(
     1,K):MERK = K
1410  NEXT K
1420  IF PIVOT = 0 THEN DET = 0:
                          RETURN
1430  REM
1440  FOR LAUF = 1 TO ZZ: REM  VE
     RTAUSCHE SPALTE 1 MIT SPALTE
      MERK
1450 HILF(LAUF,MERK) = MA(LAUF,ME
     RK)
1460 MA(LAUF,MERK) = MA(LAUF,1)
1470 MA(LAUF,1) = HILF(LAUF,MERK)

1480  NEXT LAUF
1490  REM
1500 HIDET =  - HIDET: REM  ENDE
     PIVOTSUCHE
1510  REM
1520  FOR ZE = 2 TO ZZ
1530  FOR SP = 2 TO ZZ
1540 MA(ZE,SP) = MA(1,1) * MA(ZE,
     SP) - MA(ZE,1) * MA(1,SP)
1550  NEXT SP,ZE
1560 HOCH =  SGN (MA(1,1)) *  ABS
     (MA(1,1)) ^ (ZZ - 2)
1570 HIDET = HIDET / HOCH
1580  FOR ZE = 2 TO ZZ: FOR SP =
     2 TO ZZ
1590 MA(ZE - 1,SP - 1) = MA(ZE,SP
     )
1600  NEXT SP,ZE
1610 ZZ = ZZ - 1: GOTO 1360
1620 DET = HIDET * (MA(1,1) * MA(
     2,2) - MA(1,2) * MA(2,1))
1630  RETURN : REM  ENDE DETERMIN
     ANTENBERECHNUNG
1640  REM  ----------------------
```

Beispiel 1 (BASIC)

```
]RUN
DETERMINANTE
GEBEN SIE DIE ZEILENZAHL AN: 3

UND NUN DIE MATRIXELEMENTE (ZEILENWEISE)


A(1,1)=
?0
A(1,2)=
?-1
A(1,3)=
?2
A(2,1)=
?1
A(2,2)=
?0
A(2,3)=
?1
A(3,1)=
?0
A(3,2)=
?1
A(3,3)=
?5
         BITTE WARTEN ...
DIE DETERMINANTE HAT DEN WERT 7
```

Beispiel 2 (BASIC)

```
]RUN
DETERMINANTE
GEBEN SIE DIE ZEILENZAHL AN: 3

UND NUN DIE MATRIXELEMENTE (ZEILENWEISE)


A(1,1)=
?0
A(1,2)=
?0
A(1,3)=
?0
A(2,1)=
?1
A(2,2)=
?4
A(2,3)=
?7
A(3,1)=
?-5
A(3,2)=
?1
A(3,3)=
?0
         BITTE WARTEN ...
DIE DETERMINANTE HAT DEN WERT 0
```

Pascal-Programm Determinante

```
PROGRAM DETERMINANTE;


CONST MAX 10;     (* MAXIMALE ZEILEN- UND SPALTENZAHL *)

TYPE  MATRIX = ARRAYÄ1..MAX,1..MAXÜ OF REAL;

VAR   I1        : INTEGER;
      MATA      : MATRIX;


PROCEDURE LIESMAT(L,M:INTEGER;VAR MAT:MATRIX);
   VAR     ZEILE,SPALTE : INTEGER;

   BEGIN
     FOR ZEILE:=1 TO L DO BEGIN
       FOR SPALTE:=1 TO M DO BEGIN
         WRITE('MAT(',ZEILE,',',SPALTE,')=');
         READ(MATAZEILE,SPALTEÜ)
       END;       (* OF SPALTE *)
     END;         (* OF ZEILE *)
     WRITELN;WRITELN;
   END;           (* OF LIESMAT *)


FUNCTION DET(N:INTEGER;VAR MAT:MATRIX):REAL;

   VAR  PIVOT,P,Q,HOCH,HILFSDET : REAL;
        ZEILENZAHL,ZEILE,SPALTE,MERKE,LAUF,K,L : INTEGER;
        HILF                    : MATRIX;

   BEGIN
     HILFSDET:=1;
     ZEILENZAHL:=N;

     (* UMFORMUNG DER DETERMINANTE MITTELS 'VERDICHTUNG' *)
     WHILE ZEILENZAHL>2 DO BEGIN
        IF MATÄ1,1Ü=0 THEN BEGIN   (* PIVOTSUCHE *)
          PIVOT:=0;
          FOR K:=2 TO ZEILENZAHL DO
          IF ABS(MATÄ1,KÜ)>ABS(MATÄ1,K-1Ü) THEN BEGIN
            PIVOT:=MATÄ1,KÜ;
            MERKE:=K
          END;         (* OF MAXELEMENT *)
          IF PIVOT=0 THEN BEGIN
            DET:=0;
            EXIT(DET);
          END;
          FOR LAUF:=1 TO ZEILENZAHL DO BEGIN
          (* VERTAUSCHE SPALTE 1 MIT SPALTE MERKE *)
            HILFÄLAUF,MERKEÜ:=MATÄLAUF,MERKEÜ;
            MATÄLAUF,MERKEÜ:=MATÄLAUF,1Ü;
            MATÄLAUF,1Ü:=HILFÄLAUF,MERKEÜ
          END;         (* OF VERTAUSCHE *)
          HILFSDET:=-HILFSDET;
        END;           (* OF PIVOTSUCHE *)
```

```
        (* VERDICHTUNG *)
        P:=MATÄ1,1Ü;
        FOR ZEILE:=2 TO ZEILENZAHL DO BEGIN
          Q:=MATÄZEILE,1Ü;
          FOR SPALTE:=2 TO ZEILENZAHL DO
            MATÄZEILE,SPALTEÜ:=P*MATÄZEILE,SPALTEÜ-Q*MATÄ1,SPALTEÜ
        END;            (* OF ZEILE *)
                        (* ENDE VERDICHTUNG *)
        HOCH:=MATÄ1,1Ü;
        L:=ZEILENZAHL-2;
        WHILE L>1 DO BEGIN (* BERECHNE MAT HOCH (ZEILENZAHL-2) *)
          HOCH:=HOCH*MATÄ1,1Ü;
          L:=L-1

        END;            (* OF MAT HOCH *)
        HILFSDET:=HILFSDET/HOCH;
        FOR ZEILE:=2 TO ZEILENZAHL DO BEGIN
        (* REDUZIERUNG DER MATRIXDIMENSION *)
          FOR SPALTE:=2 TO ZEILENZAHL DO
          MATÄZEILE-1,SPALTE-1Ü:=MATÄZEILE,SPALTEÜ
        END;            (* OF REDUZIERUNG *)
        ZEILENZAHL:=ZEILENZAHL-1
      END;              (* OF (ZEILENZAHL>2) OR (NOT AUSGANG) *)
      DET:=HILFSDET*(MATÄ1,1Ü*MATÄ2,2Ü-MATÄ1,2Ü*MATÄ2,1Ü);
   END;                 (* OF DET *)


BEGIN   (* HAUPTPROGRAMM *)

   WRITE(CHR(12));    (* LOESCHT BILDSCHIRM *)
   WRITELN('DETERMINANTE');
   WRITELN('============');
   WRITELN;WRITELN;
   WRITE('ZEILENZAHL ?');
   READ(I1);
   LIESMAT(I1,I1,MATA);
   WRITELN('DETERMINANTE = ',DET(I1,MATA):12:6)

END.
```

5 LITERATUR

A i t k e n, A.C.: Determinanten und Matrizen. Mannheim: Bibliographisches Institut 1969

Polynomerzeuger

von Karl Achilles

1 AUFGABENSTELLUNG

Ein Polynom P n-ten Grades habe n reelle Nullstellen, welche bekannt seien. Gesucht sind die Koeffizienten a_i (i = 1 bis n) des normierten Polynoms.

2 BESCHREIBUNG DES LÖSUNGSWEGES

2.1 Beispiel

-2; 1; 3 seien die Nullstellen.
Das normierte Polynom hat dann die Linearfaktorzerlegung

$$P = (x+2)(x-1)(x-3) = x^3-2x^2-5x+6.$$

Ergebnis $a_3=1 \quad a_2=-2 \quad a_1=-5 \quad a_o=6$

2.2 Verallgemeinerung

$x_1;\ x_2;\ \ldots;\ x_{n-1};\ x_n$ seien die n Nullstellen eines Polynoms n-ten Grades.
Es gilt dann:

$$\underline{P = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_o.}$$

Nach dem Wurzelsatz von VIETA sind die a_i mit den x_i folgendermaßen verknüpft:

$$a_n = 1$$

$$a_{n-1} = -(x_1+x_2+ \ldots +x_n)$$

$$a_{n-2} = x_1x_2+x_1x_3+x_2x_3+ \ldots +x_{n-1}x_n$$

$$a_{n-3} = -(x_1x_2x_3+x_1x_2x_4+ \ldots +x_{n-2}x_{n-1}x_n)$$

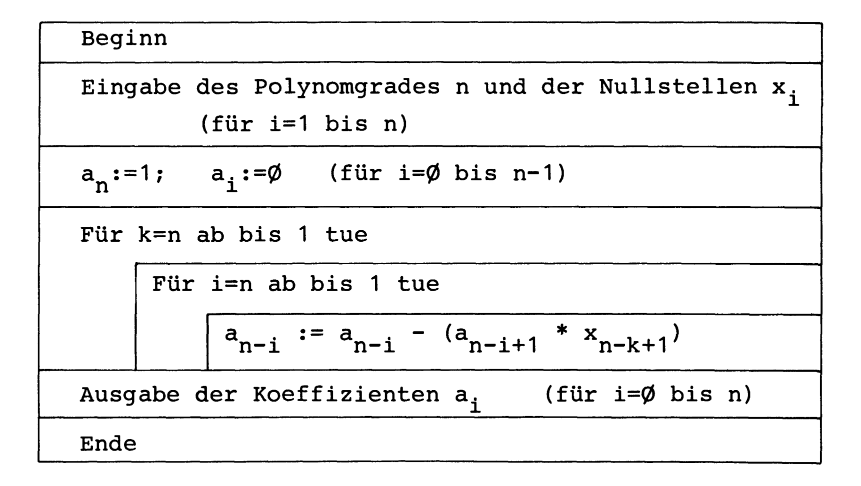
$$\vdots \qquad \vdots \qquad \vdots$$

$$a_o = (-1)^n \cdot x_1x_2x_3 \cdot \ldots \cdot x_{n-1}x_n$$

3 PROGRAMMBESCHREIBUNG

Wie das folgende Struktogramm verdeutlicht, kann man für den VIETAschen Wurzelsatz einen einfachen Algorithmus angeben.

3.1 Struktogramm



Daß dieser Algorithmus das Problem löst, wird im folgenden exemplarisch gezeigt. In der Tabelle wird der Berechnungsteil aus dem Struktogramm (für die a_i für i=Ø bis n-1) durchgespielt.

Zugrunde liegt das Beispiel: $n = 3$

$x_1 = -1$

$x_2 = 2$

$x_3 = 3$

Zu Beginn der Rechnung gilt: a_3=1 und a_i=Ø (i=Ø bis 2)

Tabelle (Belegungsplan):

k	i	a_3	a_2	a_1	a_o
3	3	1	Ø	Ø	Ø
3	2	1	Ø	Ø	Ø
3	1	1	1	Ø	Ø
2	3	1	1	Ø	Ø
2	2	1	1	-2	Ø
2	1	1	-1	-2	Ø
1	3	1	-1	-2	6
1	2	1	-1	1	6
1	1	1	-4	1	6

1. Nullstelle wird verwendet

2. Nullstelle wird verwendet

3. Nullstelle wird verwendet

Das Ergebnis ist also:

$a_3 = 1$

$a_2 = -4$

$a_1 = 1$

$a_o = 6$

3.2 Programmlistings

```
0PR#0
0LIST

1000  REM   POLYNOMERZEUGER
1010  REM   ***************
1020  REM   KARL ACHILLES 14.1.82
      ***
1030  REM
1040  HOME
1050  INVERSE : PRINT "POLYNOMERZ
    EUGER": NORMAL
1060  VTAB (5)
1070  REM
1080  REM   EINGABE
1100  INPUT "GEBEN SIE DEN GRAD D
    ES POLYNOMS AN .. ";N
1105  DIM X(N),A(N)
1110  PRINT : PRINT "GEBEN SIE NU
    N DIE NULLSTELLEN AN :"
1120  FOR I = 1 TO N
1130  PRINT I".NULLSTELLE .. ";
1132  INPUT X(I)
1135  NEXT I
1137  REM   ENDE EINGABE
1138  REM
1139  REM   BERECHNUNG DER KOEFFI
    ZIENTEN
1140  LET A(N) = 1
1150  FOR I = 0 TO N - 1
1155  LET A(I) = 0
1157  NEXT I
1160  FOR K = N TO 1 STEP  - 1
1170  FOR I = N TO 1 STEP  - 1
1180  LET A(N - I) = A(N - I) - (
    A(N - I + 1) * X(N - K + 1))

1190  NEXT 1,K
1195  REM   ENDE BERECHNUNG
1197  REM
1200  REM  AUSGABE DER A(I)
1205  PRINT : PRINT : PRINT "KOEF
    FIZIENTEN DES POLYNOMS": PRINT

1210  FOR I = N TO 0 STEP  - 1
1220  PRINT "A("I") = "A(I)
1230  NEXT 1
1250  END
```

```
PROGRAM POLYNOMERZEUGER;

CONST   MAXIMALGRADE = 30;     (* WILLKUERLICH GEWAEHLT *)

VAR     GRAD, I, K    :INTEGER;
        HILF          :REAL;
        NULLSTELLE    :ARRAYÄ1..MAXIMALGRADÜ OF REAL;
        KOEFFIZIENT   :ARRAYÄ0..MAXIMALGRADÜ OF REAL;
BEGIN
        (* UEBERSCHRIFT *)
   WRITE(CHR(12));      (* SAEUBERT BILDSCHIRM *)
   WRITELN('POLYNOMERZEUGER');
   WRITELN('---------------');
   WRITELN;WRITELN;

        (* EINGABE *)
  WRITE ('BITTE POLYNOMGRAD EINGEBEN:');
  READLN (GRAD); WRITELN;
  WRITELN ('BITTE NULLSTELLEN EINGEBEN');
  WRITELN ('---------------------------');
  WRITELN;
  FOR K := 1 TO GRAD DO
  BEGIN
    WRITE ('NULLSTELLE NR.',K,' = ');
    READLN (NULLSTELLEÄKÜ);
  END;

        (* KOEFFIZIENTENANFANGSWERTE *)
  FOR I := 0 TO GRAD-1 DO
    KOEFFIZIENTÄIÜ := 0;
  KOEFFIZIENTÄGRADÜ := 1;

        (* KOEFFIZIENTENBERECHNUNG *)
  FOR K := GRAD DOWNTO 1 DO
  BEGIN
    FOR I := GRAD DOWNTO 1 DO
    BEGIN
    HILF := NULLSTELLEÄGRAD-K+1Ü;
    KOEFFIZIENTÄGRAD-IÜ := KOEFFIZIENTÄGRAD-IÜ-KOEFFIZIENTÄGRAD-I+1Ü*HILF
    END;
  END; (* ENDE KOEFFIZIENTENBERECHNUNG *)

        (* KOEFFIZIENTEN AUSDRUCKEN *)
  FOR I := 1 TO 3 DO   WRITELN;
  FOR I := GRAD DOWNTO 0 DO
  WRITELN ('KOEFFIZIENT A(',I,') = ',KOEFFIZIENTÄIÜ);

END.
```

4 WEITERE BEISPIELE

4.1 Nullstellen -4 -2 -1 1 2

Koeffizienten $a_5=1$ $a_4=4$ $a_3=-5$ $a_2=-20$ $a_1=4$ $a_0=16$

4.2 Nullstellen 0.5 -1.5 -6.1 0.01 0

Koeffizienten $a_5=1$ $a_4=7.09$ $a_3=5.279$ $a_2=-4.6285$ $a_1=0.4575$ $a_0=0$

Größter gemeinsamer Teiler (ggT)

von Karl Achilles

1 AUFGABENSTELLUNG

Von zwei natürlichen Zahlen a,b soll der ggT bestimmt werden nach dem Euklidischen Divisionsalgorithmus.

2 LÖSUNGSWEG

Der Euklidische Divisionsalgorithmus basiert auf einer fortlaufenden Teilung mit Rest. Er terminiert, wenn der Rest Null geworden ist. Dies sei an einem Beispiel veranschaulicht:

Zu bestimmen sei ggT(437;69):

```
437 : 69  =  6 Rest 23
 69 : 23  =  3 Rest Ø
```

Also ist ggT(437;69) = 23

3 PROGRAMMBESCHREIBUNG

3.1 Struktogramm

Anfang	
Lies (A,B)	
Wiederhole	R <— A MOD B A <— B B <— R
	bis R = Ø
Schreibe (A)	
Ende	

3.2 Programmlistings

Größter gemeinsamer Teiler: BASIC-Programm

```
ÜLIST

100  REM  * GGT *
110  REM
120  HOME
130  INVERSE : PRINT "GGT": NORMAL

140  VTAB (5)
150  PRINT "GEBEN SIE BITTE ZWEI
     NATUERLICHE ZAHLEN A,B EIN :
     "
160  INPUT "A = ";A
170  INPUT "B = ";B
180  LET GGT = A: LET V = B
190  REM
200  REM  BERECHNE GGT
210  LET REST = GGT - V *  INT (G
     GT / V)
220  LET GGT = V
230  LET V = REST
240  IF REST > 0 THEN 210
250  REM
260  REM  AUSGABE
270  HOME
280  PRINT "GGT("A";"B") = "GGT
300  END
```

Beispiele in BASIC:

```
ÜRUN
GGT
GEBEN SIE BITTE ZWEI NATUERLICHE ZAHLEN A,B EIN :
A = 75
B = 15
GGT(75;15) = 15
```

```
ÜRUN
GGT
GEBEN SIE BITTE ZWEI NATUERLICHE ZAHLEN A,B EIN :
A = 87654321
B = 12345678
GGT(87654321;12345678) = 9
```

Größter gemeinsamer Teiler: Pascal-Programm

```
PROGRAM GGTEILER;

VAR  ZAHL1, ZAHL2, REST, GGT, V :INTEGER;


BEGIN   (* HAUPTPROGRAMM *)

  (* EINGABE *)
  WRITELN ('BITTE 2 NATUERLICHE ZAHLEN EINGEBEN');
  READ (ZAHL1,ZAHL2);
  GGT := ZAHL1;  V := ZAHL2;
  (* BERECHNUNG DES GGT *)

  REPEAT
    REST := GGT MOD V;
    GGT := V; V := REST
  UNTIL REST = 0;

  (* AUSDRUCKEN DES GGT *)
  WRITELN;
  WRITE ('GGT (',ZAHL1,';',ZAHL2,') = ',GGT);
END.
```

3.3 Bemerkungen zur Programmierung

Für die Division mit ganzzahligem Rest gibt es in Pascal die Modulo-Funktion (REST = GGT MOD V). Zum Beispiel ergibt sich für GGT=17 und V=5 der ganzzahlige Divisionsrest 2; also ist 17 MOD 5 = 2.

Die Modulo-Funktion muß in BASIC erst simuliert werden. Unter Verwendung der BASIC-Funktion INT (Ganzzahliger Teil einer Zahl) kann man folgende Beziehung für den ganzzahligen Divisionsrest aufstellen:

$$REST = GGT - V \cdot INT(GGT/V) \qquad (3.31)$$

Für GGT=17 und V=5 ist dann INT(GGT/V)=3 und REST=17-5·3=2. Die Beziehung (3.31) ersetzt also die Formel REST=GGT MOD V.

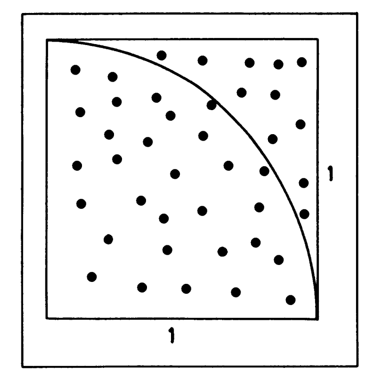
Monte-Carlo-Pi

von Karl Achilles

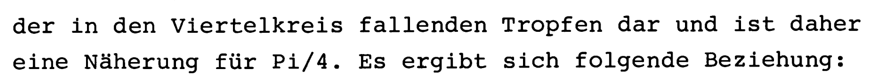
1 AUFGABENSTELLUNG

Die Kreiszahl Pi soll mit Hilfe eines Zufallsexperiments geschätzt werden (Monte-Carlo-Methode).

2 LÖSUNGSWEG



Dazu läßt man N Regentropfen auf ein Quadrat der Seitenlänge 1 fallen (siehe Skizze) und zählt die Anzahl Z der in das Innere des Viertelkreises fallenden Tropfen.
Der Flächeninhalt des Viertelkreises mit dem Radius 1 entspricht Pi/4.
Der Bruch Z/N stellt den Anteil der in den Viertelkreis fallenden Tropfen dar und ist daher eine Näherung für Pi/4. Es ergibt sich folgende Beziehung:



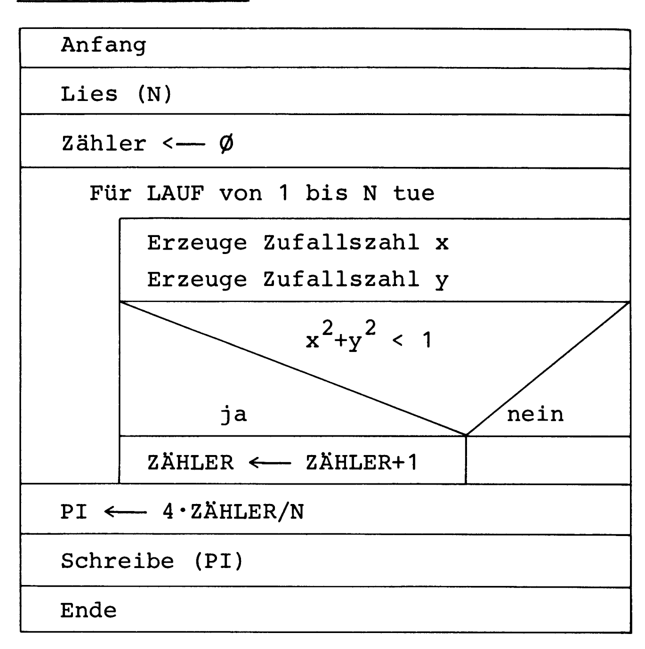
$$\text{Pi} \approx \frac{4 \cdot Z}{N}$$

Der Simulation des Zufallsexperiments liegt folgende Idee zugrunde: Man erzeugt mit Hilfe eines Zufallsgenerators hintereinander zwei Zufallszahlen x,y im Bereich zwischen Ø und 1.

Denkt man sich den Ursprung eines Koordinatensystems in die linke untere Ecke des Quadrats gelegt, so wird durch die Koordinaten x,y ein Punkt innerhalb des Quadrats beschrieben. Dieser Punkt liegt genau dann innerhalb des Viertelkreises, wenn die Bedingung $x^2 + y^2 < 1$ erfüllt ist. Alle Punkte mit den Koordinaten x,y, die auf dem Kreis liegen, erfüllen bekanntlich die Kreisgleichung $x^2 + y^2 = 1$.
Es sollte nicht unerwähnt bleiben, daß die Monte-Carlo-Methode sehr schlecht konvergiert und daß unbedingt ein guter Zufallsgenerator verwendet werden sollte!

3 PROGRAMMBESCHREIBUNG

3.1 Struktogramm



3.2 Programmlistings

Monte-Carlo-Pi: BASIC-Programm

```
üPR#0
üLIST

100  REM  * MONTE-CARLO-PI *
110  REM
120  HOME
130  INVERSE : PRINT "MONTE-CARLO
     -PI": NORMAL
140  PRINT : PRINT
150  INPUT "WIEVIELE VERSUCHE ";N

160  LET ZAEHLER = 0
170  FOR LAUF = 1 TO N
180  LET X =  RND (1): LET Y =  RND
     (1)
190  IF X * X + Y * Y < 1 THEN ZA
     EHLER = ZAEHLER + 1
200  NEXT LAUF
210  LET PI = 4 * ZAEHLER / N
220  PRINT "NAEHERUNG FUER PI = "
     PI
230  END
```

Beispiele in BASIC:

```
üRUN
MONTE-CARLO-PI


WIEVIELE VERSUCHE 10
NAEHERUNG FUER PI = 3.2
```

```
üRUN
MONTE-CARLO-PI


WIEVIELE VERSUCHE 1000
NAEHERUNG FUER PI = 3.128
```

Monte-Carlo-Pi: Pascal-Programm

```
PROGRAM MONTEPI ;

USES APPLESTUFF;    (* ENTHÄLT ZUFALLSGENERATOR *)

VAR  ZAEHLER,LAUF,MAX : INTEGER;
     X,Y,PI           : REAL;

BEGIN
  WRITE('WIEVIELE REGENTROPFEN ? ');
  READLN(MAX);
  ZAEHLER:=0;
  RANDOMIZE;
  FOR LAUF:=1 TO MAX
    DO BEGIN
         (* ERZEUGE ZUFALLSZAHLEN *)
         X:=RANDOM/32767;
         Y:=RANDOM/32767;
         IF X*X+Y*Y<1 THEN ZAEHLER:=ZAEHLER+1;
       END;
    PI:=4*ZAEHLER/MAX;
  WRITELN;
  WRITELN('NÄHERUNG FÜR PI = ',PI);
  WRITELN;
  WRITELN(MAX,' ZUFALLSVERSUCHE');
END.
```

3.3 Bemerkungen zur Programmierung

Der wesentliche Unterschied in den Programmen (BASIC, Pascal) besteht in der Erzeugung der Zufallszahlen:
In Pascal wird durch die Funktion RANDOM eine ganzzahlige Zufallszahl zwischen Ø und 32767 erzeugt, welche mittels Division durch 32767 auf das Intervall [Ø;1] transformiert werden muß. In BASIC steht durch die Funktion RND(1) sofort der gewünschte Bereich zur Verfügung.

4 LITERATUR

E n g e l, A.: Elementarmathematik vom algorithmischen Standpunkt, Klett 1977

D'Hondtsches Höchstzahlverfahren

von Karl Achilles

1 AUFGABENSTELLUNG

Nach einer Wahl sollen M Mandate auf N Parteien nach dem D'Hondtschen Verfahren verteilt werden. Dabei werden die Zweitstimmenzahlen jeder Partei nacheinander durch 1,2,3,... dividiert und anschließend die Mandate in der Reihenfolge der Größe der anfallenden Quotienten vergeben. Hat eine Partei weniger als 5% aller Stimmen bekommen, so erhält sie kein Mandat.

2 LÖSUNGSWEG

Für die Zweitstimmenzahlen der N Parteien wird ein Feld Z_1, Z_2, Z_3, ... ,Z_N bereitgestellt. Entsprechendes gilt für die Quotienten Q_i und die Divisoren (Teiler) T_i (i=1 bis N).
Zunächst werden die Zweitstimmenzahlen den Quotienten zugeordnet, wobei alle Teiler den Wert 1 erhalten. Es gilt also:
$Q_i = Z_i$ für i von 1 bis N. Befindet sich das Maximum aller Quotienten an der k-ten Stelle, so wird der k-ten Partei das erste Mandat zugeteilt. Der Quotient Q_k wird gestrichen und durch einen neuen Quotienten ersetzt, welcher sich aus $Q_k=Z_k/T_k$ mit $T_k=2$ ergibt. Somit hat sich für die k-te Partei der Teiler um 1 erhöht, nachdem ihr ein Mandat zugeteilt worden war.
Nun wird das nächste Maximum der Q_i (i=1 bis N) gesucht. Nachdem ein Mandat zugeteilt wurde, muß immer der entsprechende Teiler um 1 erhöht und der neue Quotient gebildet werden. Dieser ersetzt dann den vorherigen Quotienten.
Nach jeder Zuteilung wird der Mandatszähler (der alle Mandate zählt) erhöht. Das Programm bricht mit der Berechnung ab, wenn gilt: Mandatszähler = M.

3 PROGRAMMBESCHREIBUNG

3.1 Struktogramm

```
Anfang
Lies (PARTEIENANZAHL, MANDATSANZAHL)
Für I von 1 bis PARTEIENANZAHL tue
        Lies (Parteiname(I))
        Lies (ZWEITSTIMMENZAHL(I))
Berechne die GESAMTSTIMMENZAHL
Berücksichtige die 5%-Klausel
Für I von 1 bis PARTEIENANZAHL tue
        MANDATZAHL(I)  <—— Ø
        TEILER(I)      <—— 1
        QUOTIENT(I)    <—— ZWEITSTIMMENZAHL(I)
MANDATZUTEILUNG <—— Ø
Wiederhole
        MANDATZUTEILUNG <—— MANDATZUTEILUNG+1
        MAXIMUM <—— QUOTIENT(1)
        Für I von 2 bis PARTEIENANZAHL tue
                QUOTIENT(I) > MAXIMUM
                ja:   MAXIMUM <—— QUOTIENT(I)
                nein: (leer)
        Erhöhe MANDATZAHL der Partei mit MAXIMUM
        Erhöhe TEILER        "    "    "    "
        Berechne QUOTIENT    "    "    "    "
bis MANDATZUTEILUNG = MANDATSANZAHL
Für I von 1 bis PARTEIENANZAHL tue
        Schreibe (PARTEINAME(I))
        Schreibe (MANDATZAHL(I))
Ende
```

3.2 Programmlistings

D'Hondtsches Verfahren: BASIC-Programm

```
ÜC
ÜLIST

1000  REM  *** -- D'HONDT -- ***
1010  REM
1020  REM  KARL ACHILLES 9.11.81
1030  REM
1040  REM  ---------------------
     -
1050  REM  Z(I) ... ZWEITSTIMMENZ
     AHLEN
1060  REM  N$(I)... PARTEINAMEN
1070  REM  Q(I) ... QUOTIENTEN DE
     RI-TEN PARTEI
1080  REM  K(I) ... TEILER FUER D
     IE I-TE PARTEI
1090  REM  M(I) ... ANZAHL DER MA
     NDATE FUER DIE I-TE PARTEI
1100  REM  ---------------------
     -
1110  REM
1120  REM  --------------------
1130  REM     HAUPTPROGRAMM
1140  REM  --------------------
1150  REM
1160  HOME
1170  PRINT "D'HONDTSCHES VERFAHR
     EN"
1180  PRINT "===================
     =="
1190  REM
1200  GOSUB 1290: REM  EINGABE
1210  GOSUB 1460: REM  GESAMTSTIM
     MEN
1220  GOSUB 1570: REM  5%-KLAUSEL

1230  GOSUB 1650: REM  INITIALISI
     ERUNG
1240  GOSUB 1740: REM  BERECHNE M
     ANDATSVERTEILUNG
1250  GOSUB 1960: REM  AUSGABE
1260  END
1270  REM  ENDE HAUPTPROGRAMM
1280  REM
1290  REM
1300  REM  --- EINGABE ---
1310  REM  ---------------
1320  VTAB (5)
1330  INPUT "GEBEN SIE DIE ANZAHL
     DER PARTEIEN AN ";N
1340  INPUT "GEBEN SIE DIE ANZAHL
     ALLER MANDATE AN";MA
1350  DIM Z(N),N$(N),Q(N),K(N),M(
     N)
1360  VTAB (15)
1370  PRINT "LESEN SIE DIE NAMEN
     DER PARTEIEN EIN -"
1380  PRINT "ZUERST DIE PARTEI MI
     T DER HOECHSTEN STIMMENZAHL"
     : PRINT
1390  FOR I = 1 TO N: INPUT N$(I)
     : NEXT I: PRINT
1400  HOME
1410  PRINT "GEBEN SIE DIE ZWEITS
     TIMMENZAHLEN DER PARTEIEN EI
     N :": PRINT
1420  FOR I = 1 TO N: PRINT "ZWEI
     TSTIMMENZAHL DER PARTEI "N$(
     I): INPUT Z(I): NEXT I
1430  RETURN
1440  REM  ---   ENDE EINGABE ---

1450  REM
1460  REM  --- GESAMTSTIMMEN ---
1470  REM  ---------------------
1480  LET G = 0
1490  FOR I = 1 TO N: LET G = G +
     Z(I): NEXT I
1500  HOME
1510  PRINT "DIE SUMME DER ZWEITS
     TIMMEN IST: "G
1520  PRINT "-------------------
     -------------------"
1530  RETURN
1540  REM
1550  REM  --- ENDE GESAMTSTIMMEN
     ---
1560  REM
1570  REM  --- BERUECKSICHTIGUNG
     DER 5%-KLAUSEL ---
1580  REM  ---------------------
     -
1590  LET LIMIT = .05 * G
1600  FOR J = 1 TO N: LET Z(J) =
     Z(J) *  SGN ( INT (Z(J) / LI
     MIT)): NEXT J
1610  RETURN
```

D'Hondtsches Verfahren: BASIC-Beispiel

```
1620  REM
1630  REM  --- ENDE 5%-KLAUSEL --
-
1640  REM
1650  REM  --- INITIALISIERUNG --
-
1660  REM  ----------------------
-
1670  LET SM = 0
1680  FOR I = 1 TO N: LET K(I) =
     1: NEXT I
1690  FOR I = 1 TO N: LET Q(I) =
     Z(I): NEXT I
1700  RETURN
1710  REM  --- ENDE INITIALISIERU
     NG ---
1720  REM
1730  REM
1740  REM  --- BERECHNE MANDATSVE
     RTEILUNG ---
1750  REM  ----------------------
-
1760  REM
1770  LET MERKE = 1
1780  LET HILF = Q(1)
1790  FOR I = 1 TO N
1800  IF Q(I) <  = HILF THEN  GOTO
     1840: REM  VERGLEICH DER Q(I
     ) MIT Q(1)
1810  REM  ELSE ... VERTAUSCHE
1820  LET H = Q(I)
1830  LET MERKE = I
1840  NEXT I
1850  LET M(MERKE) = M(MERKE) + 1
     : REM  TEILE MANDAT DER PART
     EI NR.MERKE ZU
1860  LET SM = SM + 1: REM  SM IS
     T ZAEHLER FUER DIE ZUGETEILT
     EN MANDATE
1870  IF SM = (MA) THEN  GOTO 192
     0
1880  REM  ELSE ... ERHOEHE TEILE
     R,BILDE NEUEN QUOTIENTEN UND
      DURCHLAUFE WEITER DIE SCHLE
     IFE
1890  LET K(MERKE) = K(MERKE) + 1

1900  LET Q(MERKE) = Z(MERKE) / K
     (MERKE)
1910  GOTO 1770
1920  RETURN
1930  REM  --- ENDE BERECHNUNG --
-
1940  REM
1950  REM
1960  REM  --- AUSGABE ---
1970  REM  ---------------
1980  VTAB (10)
1990  PRINT "DIE MANDATE VERTEILE
     N SICH AUF DIE"
2000  PRINT "EINZELNEN PARTEIEN W
     IE FOLGT :"
2010  PRINT "===================
     ===================": PRINT
     : PRINT
2020  FOR I = 1 TO N
2030  PRINT  TAB( 1)N$(I); TAB( 1
     6)M(I); TAB( 21)"MANDATE"
2040  NEXT I
2050  RETURN
2060  REM  --- ENDE AUSGABE ---
```

```
]RUN
D'HONDTSCHES VERFAHREN
======================
GEBEN SIE DIE ANZAHL DER PARTEIEN AN 3
GEBEN SIE DIE ANZAHL ALLER MANDATE AN78
LESEN SIE DIE NAMEN DER PARTEIEN EIN -
ZUERST DIE PARTEI MIT DER HOECHSTEN STIMMENZAHL

?CDU
?SPD
?FDP

GEBEN SIE DIE ZWEITSTIMMENZAHLEN DER PARTEIEN EIN :

ZWEITSTIMMENZAHL DER PARTEI CDU
?987654
ZWEITSTIMMENZAHL DER PARTEI SPD
?894566
ZWEITSTIMMENZAHL DER PARTEI FDP
?190106
DIE SUMME DER ZWEITSTIMMEN IST: 2072326
---------------------------------------
DIE MANDATE VERTEILEN SICH AUF DIE
EINZELNEN PARTEIEN WIE FOLGT :
=======================================

CDU            37   MANDATE
SPD            34   MANDATE
FDP            7    MANDATE
```

D'Hondtsches Verfahren: Pascal-Programm

```
PROGRAM DHONDT;

(* DHONDTSCHES HOECHSTZAHLVERFAHREN *)

CONST   MAX = 10;    (* MAXIMALE ANZAHL DER PARTEIEN *)

TYPE    GANZBEREICH  = ARRAYÄ1..MAXÜ OF INTEGER;
        REELLBEREICH = ARRAYÄ1..MAXÜ OF REAL;
        NAMENBEREICH = ARRAYÄ1..MAXÜ OF STRING;
        PARTEIEN     = RECORD
                          PNAME        : NAMENBEREICH;
                          STIMMENZAHL,
                          TEILER,
                          QUOTIENT     : REELLBEREICH;
                          MANDATZAHL   : GANZBEREICH;
                       END;    (* OF PARTEIEN *)

VAR     PANZAHL, MANDATE, ZEIGER,LAUF : INTEGER;
        ALLE,LIMIT                    : REAL;
        PARTEI                        : PARTEIEN;

PROCEDURE LOESCH;
   VAR K : INTEGER;
   BEGIN
      FOR K:=1 TO 1000 DO BEGIN END;
      WRITE(CHR(12));
   END;    (* OF LOESCHEN *)

PROCEDURE EINGABE;
   BEGIN
     LOESCH;
     GOTOXY(10,10);
     WRITE('D''HONDTSCHES VERFAHREN');
     GOTOXY(10,12);
     WRITE('=====================');
     LOESCH;
     GOTOXY(1,10);
     WRITE('WIEVIELE PARTEIEN ?');
     GOTOXY(30,10);
     READLN(PANZAHL);
     GOTOXY(1,13);
     WRITE('WIEVIELE MANDATE INSGESAMT ? ');
     GOTOXY(30,13);
     READLN(MANDATE);
   END;           (* OF EINGABE *)

PROCEDURE LIES;
   PROCEDURE LIESSTIMMEN;
      VAR    I2 : INTEGER;

   BEGIN   (* LIESSTIMMEN *)
     WITH PARTEI
       DO BEGIN
            FOR I2:=1 TO PANZAHL DO
            BEGIN
              GOTOXY(0,I2);
              WRITE('ZWEITSTIMMENZAHL DER PARTEI ',PNAMEÄI2Ü,' ');
              GOTOXY(35,I2);
              READ(STIMMENZAHLÄI2Ü);
            END;
          END;
   END;           (* OF LIESSTIMMEN *)
```

```
   BEGIN   (* LIES *)
     LOESCH;
     FOR LAUF:=1 TO PANZAHL
       DO BEGIN
            GOTOXY(1,LAUF);
            WRITE('NAME DER ',LAUF,' . PARTEI ? ');
            GOTOXY(30,LAUF);
            READLN(PARTEI.PNAMEÄLAUFÜ);
          END;
     LOESCH;
     LIESSTIMMEN;
   END;              (* OF LIES *)

PROCEDURE ALLESTIMMEN;
   VAR     I3 : INTEGER;
   BEGIN
     LOESCH;
     ALLE:=0;
     FOR I3:=1 TO PANZAHL
       DO ALLE:=ALLE+PARTEI.STIMMENZAHLÄI3Ü;
     WRITELN;
     WRITELN('GESAMTSTIMMENZAHL = ',ALLE);
   END;            (* OF ALLESTIMMEN *)

PROCEDURE FUENFPROZENT;
   VAR     I4 : INTEGER;
   BEGIN
     LIMIT:=0.05*ALLE;
     FOR I4:=1 TO PANZAHL
       DO IF PARTEI.STIMMENZAHLÄI4Ü < LIMIT THEN PARTEI.STIMMENZAHLÄI4Ü:=Ø;
   END;            (* OF FUENFPROZENT *)

PROCEDURE INITIALISIERE;
   VAR     I5 : INTEGER;
   BEGIN
     FOR I5:=1 TO PANZAHL
       DO WITH PARTEI
         DO BEGIN
              MANDATZAHLÄI5Ü:=0;
              TEILERÄI5Ü:=1;
              QUOTIENTÄI5Ü:=STIMMENZAHLÄI5Ü;
            END;
   END;            (* OF INITIALISIERE *)

PROCEDURE BERECHNE;
   VAR     I6, MANDATZUTEILUNG, ZEIGER : INTEGER;
           MAXIMUM                     : REAL;
   BEGIN
     WITH PARTEI
       DO BEGIN
            MANDATZUTEILUNG:=0;
            REPEAT
              MANDATZUTEILUNG:=MANDATZUTEILUNG+1;
              ZEIGER:=1;
              MAXIMUM:=QUOTIENTÄ1Ü;
              FOR I6:=2 TO PANZAHL DO
                IF QUOTIENTÄI6Ü > MAXIMUM THEN
                BEGIN
                  ZEIGER:=I6;
                  MAXIMUM:=QUOTIENTÄI6Ü
                END;     (* OF IF *)
              MANDATZAHLÄZEIGERÜ:=MANDATZAHLÄZEIGERÜ+1;
              (* NEUES MANDAT ZUGETEILT *)
              TEILERÄZEIGERÜ:=TEILERÄZEIGERÜ+1;
              (* TEILER ERHOEHT *)
```

```
              QUOTIENTAZEIGERÜ:=STIMMENZAHLÄZEIGERÜ/TEILERÄZEIGERÜ;
              (* NEUEN QUOTIENTEN BERECHNET *)
            UNTIL MANDATZUTEILUNG = MANDATE;
          END;
   END;            (* OF BERECHNE *)

PROCEDURE AUSGABE;
   VAR     I7 : INTEGER;
   BEGIN
     WRITE(CHR(12));
     GOTOXY(5,5);
     WRITELN('MANDATSVERTEILUNG :');
     GOTOXY(5,7);
     WRITELN('==================');
     FOR I7:=1 TO PANZAHL
       DO BEGIN
            LAUF:=I7+10;
            GOTOXY(1,LAUF);
            WRITE(PARTEI.PNAMEÄI7Ü);
            GOTOXY(20,LAUF);
            WRITE(PARTEI.MANDATZAHLÄI7Ü);
            GOTOXY(25,LAUF);
            WRITE('MANDATE');
          END;
   END;            (* OF AUSGABE *)


BEGIN              (* HAUPTPROGRAMM *)
   EINGABE;
   LIES;
   ALLESTIMMEN;
   LOESCH;
   GOTOXY(10,10);
   WRITE('BITTE WARTEN ...');
   FUENFPROZENT;
   INITIALISIERE;
   BERECHNE;
   AUSGABE;
END.
```

3.3 Bemerkungen zur Programmierung

Einer Partei, die weniger als 5% der Gesamtstimmenzahl erreicht, wird die Zweitstimmenzahl Ø zugewiesen. Dadurch ergeben sich alle Quotienten zu Null, und die Partei erhält kein Mandat.
In der Pascal-Version werden alle Daten einer Partei mit Hilfe des Datentyps RECORD (Verbund) zusammengefaßt. Dieser Typ ermöglicht den Zugriff auf sämtliche Daten einer Partei mit einer einzigen Variablen.
Das N-Komponenten-Feld wird in Pascal mit ARRAY[1..MAX] OF Typ bereitgestellt, während in BASIC z.B. die Anweisung DIMZ(N) notwendig ist.

Unterprogramme werden in Pascal im Deklarationsteil als PROCEDURES aufgeführt (Blockstruktur), in BASIC stehen die SUBROUTINES hinter dem Hauptprogramm. Sie werden mit GOSUB Zeilennummer angesprungen und enden mit dem Rücksprungbefehl RETURN.